UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Centre de Formació Interdisciplinària Superior

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

telecos BCN

Degree in Mathematics

Degree in Engineering Physics

Bachelor's Degree Thesis

Learning graphs from data and spectral clustering with applications to finance

Josep M Gallegos Saliner

May 2019

Supervised by Daniel P. Palomar

Tutored by Jordi Castro

Department of Electronic & Computer Engineering

Hong Kong University of Science and Technology

First of all, thanks to Prof. Daniel P. Palomar and Dr. Sandeep Kumar for receiving me in their research group and being so attentive and helpful to every doubt that arise.

Thanks to Fundació Cellex and CFIS for giving me the support and help with the mobility scholarship, allowing me to take part in this international research experience. Thanks to Genís and Alejandro for sharing this time and all the help given. Thanks to my mother for all her support.

And to finish, special thanks to Alexandra for all the interest and selfless advice she gave me.

Abstract

Graph representations of data sets give us many possibilities to enhance the processing, analysis and visualization of data. Specifically, they are fundamental for spectral clustering: a family of algorithms that separate the data points of a set into groups by using the spectrum of its graph representation. As these representations are not unique, we give in this bachelor's thesis a survey of methods useful to learn them. Some approaches are geared towards the general setting but others try to take advantage of specific properties of certain data sets. We also study the family of spectral clustering algorithms with a theoretical approach but also with a focus on its performance.

As a practical application, these techniques are used to find clusters in the stock market with the aim of comparing the structure of the clusters with the industry classification of companies. Finally, we have used these representation techniques to visualize the increase of systemic risk associated with an economic crisis.

Keywords

Spectral clustering, graph Laplacian, graph learning

Contents

1. Introduction

Clustering a data set consists in finding a partition of its data points such that the elements belonging to the same set are similar in some way. The aim of clustering is to explore and find structure in data without any human guidance, which makes it especially important nowadays with the large quantities of data available.

Some direct applications of clustering are in market segmentation, identifying groups of people who might be susceptible to purchase a specific product, or in image segmentation, finding a partition of a digital image into groups that represent different elements in the image.

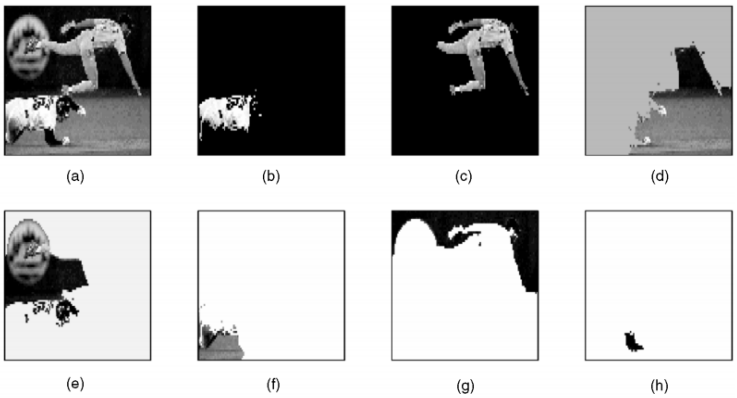

Figure 1: Example of image segmentation[18]. In (a) there is the initial image and in (b), ..., (h), the different objects detected using spectral clustering.

In this thesis, we will study specifically spectral clustering, which is a very powerful and simple to implement method for clustering data points. However, it requires a graph representation of the data set. Such a graph representation is usually not available outside the study of some specific field like social networks. Nevertheless, it is also very interesting by itself since it gives rise to new visualization and analysis possibilities. These representations are not unique and well defined, so we consider different kinds of graphs:

- For general data sets that only need to be in a metric space, we consider representations that only depend on the distance between the different points.
- For data sets that consist of collections of i.i.d. random variables, we also consider representations depending on the covariance matrix.

As a practical application we study the stock market in two different ways:

- Finding a clustering of the stocks based only on its daily returns with the aim to create portfolios robust to certain market downfalls.
- Using the graph visualization to observe a change in the number of connections during the subprime mortgage crisis of 2008.

This bachelor's thesis is divided into four sections. The first is organized as a tutorial or introduction to the different aspects of spectral clustering. It starts with the necessary definitions, gives some intuition on

why it works and finally ends with a comparison of performances of different algorithms. There are some extra topics, which are not crucial, presented in the annex. These are about giving fast approximations for spectral clustering (in Annex A) and its relationship with k-means (in Annex B). The second part explains four different approaches to estimate a graph with different properties from the covariance matrix. The third one establishes the models for the returns of the stock market and studies different ways to clear up a possible problem with learning the graphs under the presence of an undesired factor. These approaches are presented with the aim to try to refine the estimation of the factor. In the last part, we apply all the different graph learning methods to the stock market and compare the different clusterings obtained with each graph. Since the aim of this thesis is not to study accurately the stock market, the conclusions of the experiments should be taken skeptically.

2. Spectral Clustering

The reference for most of this first part on spectral clustering comes from the tutorial by Luxburg[20] which is a very comprehensive tutorial on the topic.

2.1 Basic clustering

Clustering a data set consists in finding a partition of its points such that the elements belonging to the same set are similar in some way and the elements belonging to different sets are not. The concept of similarity is not something unique, it depends on the problem to solve. However, we generally assume that two points are similar if they are close to each other.

K-means algorithm

Given n points in $\mathbb{R}^m$, we want to find a partition of the points in k sets $C_1, \ldots, C_k$ such that the points in a set are close between them and far from the ones in other sets. To get this, we may try to minimize within-cluster variation of the sets.

Definition 2.1. The within-cluster variation of a set C_j is:

$$\mathrm{WCV}(C_j) = \frac{1}{|C_j|} \sum_{x_i, x_l \in C_j} \|x_i - x_l\|_2^2$$

where x_i and x_l are points in the set.

The problem we want to solve is:

$$\underset{C_1,\ldots,C_k}{\text{minimize}} \quad \sum_{j=1}^{k} \frac{1}{|C_j|} \sum_{x_i, x_l \in C_j} \|x_i - x_l\|_2^2$$

This function is hard to minimize, but we can observe that:

$$\frac{1}{|C_j|} \sum_{x_i, x_l \in C_j} \|x_i - x_l\|_2^2 = 2 \sum_{x_i \in C_j} \|x_i - \bar{x}_j\|_2^2$$

where $\bar{x}_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$ is the centroid of the cluster C_j. We can find a local minimum with the following iterative algorithm:

1. Assign a number between 1 and k randomly to each point. This is their initial cluster.
2. Repeat iteratively until the clusters stop changing:
   (a) For each cluster compute its centroid.
   (b) For each point assign it to the cluster with the nearest centroid.

This algorithm has the problem that the resulting clusters form convex sets in the space. This can lead to issues when the ideal clusters form non convex sets.

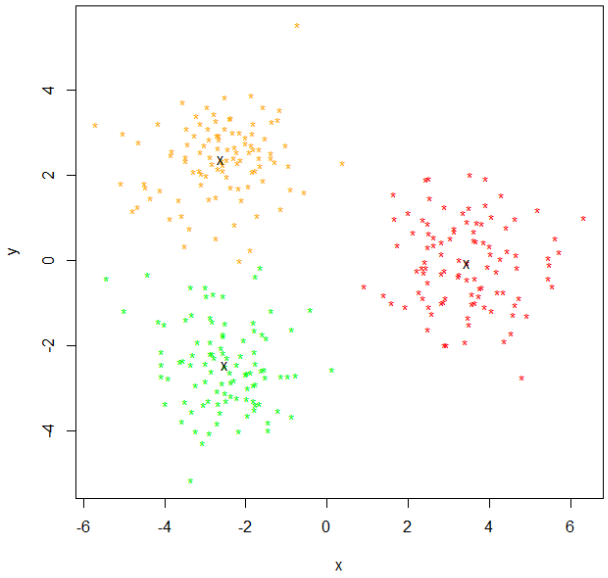

Figure 2: Example of a data set clustered using k-means with $k = 3$. In black there are the centroids of each cluster.

2.2 Graph notation

In this thesis we will only consider undirected weighted graphs $G = (V, E, w)$ without self loops and with $|V| = n$.

Definition 2.2. The weight matrix $\mathbf{W}$ of an undirected weighted graph is defined as: $\mathbf{W}_{ij} := w(v_i v_j) \geq 0$. It is a symmetric matrix that uniquely identifies the graph structure. The graphs without self loops also satisfy: $\mathbf{W}_{ii} = 0,\ i = 1 \dots n$.

Definition 2.3. The degree d_i of the vertex v_i is the sum of the weights of its incident edges: $d_i := \sum_{v_j \in V} w(v_i v_j),\ i = 1 \dots n$.

Definition 2.4. The degree matrix of a weighted graph is the diagonal matrix $\mathbf{D}$ that contains the degrees $d_1, \dots, d_n$ of the vertices on the diagonal.

Graph Laplacians

The graph Laplacian is an alternative matrix representation of a weighted graph which presents many similarities with the Laplacian operator.

Definition 2.5. We define the unnormalized Laplacian of a graph as: $\mathbf{L} := \mathbf{D} - \mathbf{W}$.

Definition 2.6. The symmetric normalized Laplacian of a graph is defined as: $\mathbf{L}_{sym} := \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$.

These matrix representations satisfy certain important properties:

Theorem 2.7. *The unnormalized graph Laplacian and the symmetric normalized graph Laplacian satisfy:*

1. *They are symmetric and positive semi-definite.*

2. *The multiplicity of the eigenvalue 0 is equal to the number of connected components $A_1, ...A_k$.*

3. *In the case of the unnormalized Laplacian , the kernel is spanned by the indicator vectors $\mathbf{1}_{A_1}, ..., \mathbf{1}_{A_k}$ of those components.*

Proof. The symmetry comes from the symmetry of both $\mathbf{D}$ and $\mathbf{W}$. To see that it is positive definite:

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \mathbf{f}^T\mathbf{D}\mathbf{f} - \mathbf{f}^T\mathbf{W}\mathbf{f} = \sum_i d_i f_i^2 - \sum_{i,j} w_{ij} f_i f_j = \sum_{i,j} w_{ij}(f_i^2 - f_i f_j) =$$

$$= \sum_{i,j} w_{ij} \left(\frac{f_i^2 + f_j^2}{2} - f_i f_j \right) = \sum_{i,j} \frac{1}{2} w_{ij}(f_i - f_j)^2 \geq 0$$

since $w_{ij} \geq 0$ and the other term is a square.

If $\mathbf{f}$ is the indicator vector of the component A_t for the unnormalized Laplacian:

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \frac{1}{2}\sum_{i,j} w_{ij}(f_i - f_j)^2 = \frac{1}{2}\sum_{v_i, v_j \in A_t} w_{ij}(1-1)^2 + \frac{1}{2}\sum_{v_i \in A_t, v_j \notin A_t} w_{ij}(1-0)^2$$

$$+\frac{1}{2}\sum_{v_i \notin A_t, v_j \in A_t} w_{ij}(0-1)^2 + \frac{1}{2}\sum_{v_i \notin A_t, v_j \notin A_t} w_{ij}(0-0)^2 = \frac{1}{2}\sum_{v_i \in A_t, v_j \notin A_t} w_{ij}(1-0)^2$$

$$+\frac{1}{2}\sum_{v_i \notin A_t, v_j \in A_t} w_{ij}(0-1)^2$$

But observe that $w_{ij} = 0$ if $v_i \in A_t$ and $v_j \notin A_t$ or vice versa since they belong on disconnected components. Thus the result is 0.

In the other direction, if $\mathbf{f}$ is an eigenvector of eigenvalue 0 of $\mathbf{L}$, whenever $w_{ij} \neq 0$ (v_i and v_j are connected), f_i must be equal to f_j. Hence $\mathbf{f}$ is constant in a connected component of the graph. Thus, $\mathbf{f}$ is a linear combination of the indicator vectors of the components. □

There are many other graph Laplacians such as:

Definition 2.8. We define the random walk normalized Laplacian of a graph as: $\mathbf{L}_{rw} := \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$.

Definition 2.9. Let $\mathbf{J}$ be the matrix that contains a one in every entry: $\mathbf{J} := \mathbf{1}\mathbf{1}^T$. We define the regularized weight matrix of a graph as: $\mathbf{W}_\tau := \mathbf{W} + \tau(\mathbf{J} - \mathbf{I})$ for $\tau \geq 0$ and the regularized degree matrix $\mathbf{D}_\tau$ of a graph in a similar way.

Definition 2.10. We define the regularized normalized Laplacian of a graph as: $\mathbf{L}_\tau := \mathbf{I} - \mathbf{D}_\tau^{-1/2}\mathbf{W}_\tau\mathbf{D}_\tau^{-1/2}$.

2.3 Basic similarity measures

To get a graph structure out of a data set, we will consider the multidimensional data points as vertices but we need to choose which pairs of points are connected and which is the weight of their connection. There is no "correct" way to do this so we will consider only some easy to interpret methods. We would like, in general, connected graphs or, at least, without isolated vertices. The presence of isolated vertices difficults the clustering task as they should belong to their own cluster with a single point.

The following methods give us a way to choose the edges that will be present in the graph. If their weight is not mentioned, we will assume that it is 1.

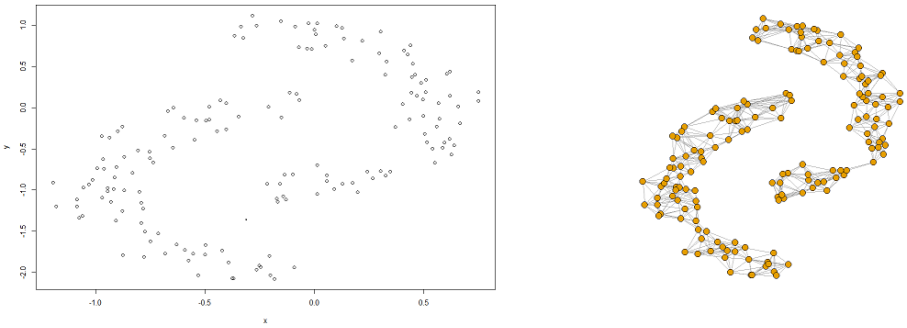

Figure 3: Example of a data set in the plane and some possible graph representation learned using k-nearest neighbors.

ϵ-neighborhood graph

Set two vertices v_i and v_j as connected if the corresponding data points satisfy: $\|\mathbf{x}_i - \mathbf{x}_j\|_2 < \epsilon$. The parameter $\epsilon > 0$ controls the size of the neighborhoods between points. We can guarantee that the graph is connected by choosing ϵ with the following algorithm:

1. Consider the complete graph between the points with weights set as the euclidean distance between points.
2. Compute the minimum spanning tree of this graph.
3. Set ϵ equal to the weight of the edge with maximum weight of the MST.

k-nearest neighbors graph

Set two vertices v_i and v_j as connected if $\mathbf{x}_i$ is one of the k closest points to $\mathbf{x}_j$ or vice versa. The parameter k controls the size of the neighborhoods between points. The graphs obtained with this method are usually sparse which is a very useful property, both computationally and for visualization.

Gaussian similarity graph

Set every pair of points as connected but with the following weight $\mathbf{W}_{ij} := \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/(2\sigma^2)\right)$. Here the parameter σ controls the size of the neighborhoods between points.

Self tuning Gaussian similarity graph

Set every pair of points as connected but with the following weight $\mathbf{W}_{ij} := \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/(2\sigma_i\sigma_j)\right)$. This way we do not have a single σ which allows us to have neighborhoods with different sizes. This is useful when the different clusters have very distinct densities or are in diverse scales. One way [21] to determine σ_i is setting it as the distance of $\mathbf{x}_i$ with its seventh closest neighbor.

Jordan estimation

We may also learn a weight matrix using convex optimization [15] such that high weight values correspond to smaller distance $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$. If we do not ask for any symmetry the problem can be solved for each row i $i = 1 \dots n$ separately:

$$\underset{\mathbf{w}_i^T\mathbf{1}=1, w_i \geq 0, w_{ii}=0}{\text{minimize}} \sum_{j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 w_{ij} + \gamma \sum_{j=1}^{n} w_{ij}^2$$

where the last term corresponds to l_2-regularization and we ask that each row adds up to one to fix a scale. But instead of choosing γ ourselves, we can choose the maximal γ such that the optimal solution has exactly m nonzero values. This method gives us an easy to compute solution and sparse graphs that only depend on the number of neighbors for every point m. It has the downfall that the result is not symmetric so we will need to symmetrize it afterwards.

Derivation of the algorithm

Let's define $e_{ij} := \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ and $\mathbf{e}_i = (e_{i1}, e_{i2}, \dots)^T$ and we can check that we can rewrite the problem as:

$$\min_{\mathbf{w}_i^T\mathbf{1}=1, \mathbf{w}_i \geq 0, w_{ii}=0} \frac{1}{2}\|\mathbf{w}_i + \frac{\mathbf{e}_i}{2\gamma}\|_2^2$$

The Lagrangian function is:

$$\mathcal{L}(\mathbf{w}_i, \eta, \beta_i) = \frac{1}{2}\|\mathbf{w}_i + \frac{\mathbf{e}_i}{2\gamma}\|_2^2 - \eta(\mathbf{w}_i^T\mathbf{1} - 1) - \beta_i^T\mathbf{w}_i$$

where $\beta_i \geq \mathbf{0}$ and η are Lagrangian multipliers.

The optimal solution should satisfy that the gradient of the Lagrangian is 0:

$$\hat{\mathbf{w}}_i + \frac{\mathbf{e}_i}{2\gamma} - \eta\mathbf{1} - \beta_i = \mathbf{0}$$

Using other KKT conditions ($w_{ij}\beta_{ij}$ should also be 0) the optimal solution $\hat{\mathbf{w}}$ satisfies:

$$\hat{w}_{ij} = \left(-\frac{e_{ij}}{2\gamma} + \eta\right)_+$$

besides for $\hat{\mathbf{w}}_{ii}$ that is 0. Now let's consider the maximal γ such that the solution has exactly m nonzero values:

- Let the m smallest components of e_i be $e_{i,k_1}, \dots, e_{i,k_m}$ (besides e_{ii}). Then $\hat{w}_{i,k_j} > 0$ for $j = 1 \dots m$ and for the others is 0.

- Considering the constraint $\mathbf{w}_i^T \mathbf{1} = 1$ we get:

$$\eta = \frac{1}{m} + \frac{1}{2m\gamma} \sum_{j=1}^{m} e_{i,k_j}$$

- We end up getting that the maximal γ such that $\hat{\mathbf{w}}_i$ is:

$$\gamma = \frac{m}{2} e_{i,k_{m+1}} - \frac{1}{2} \sum_{j=1}^{m} e_{i,k_j}$$

- Finally $\hat{w}_{i,k_j}$ for $j = 1 \dots m$:

$$\hat{w}_{i,k_j} = \frac{e_{i,k_{m+1}} - e_{i,k_j}}{m e_{i,k_{m+1}} - \sum_{h=1}^{m} e_{i,k_h}}$$

  and for the others is 0.

Is important to remember that the result is non-symmetric. In our case we will just symmetrize the result by doing: $\left(\mathbf{W} + \mathbf{W}^T\right)/2$.

Algorithm

1. Compute $e_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$.
2. For each row of $\mathbf{E}$ find the $m+1$ smallest component (without the diagonal term).
3. For each row compute: $D_i = m e_{i,k_{m+1}} - \sum_{h=1}^{m} e_{i,k_h}$
4. For each element of $\hat{\mathbf{W}}$:

$$\hat{w}_{i,j} = \frac{e_{i,k_{m+1}} - e_{i,j}}{D_i}$$

5. Keep only the positive elements of $\hat{\mathbf{W}}$ and set the others (and the diagonal) as 0.
6. Symmetrize $\hat{\mathbf{W}}$:

$$\hat{\mathbf{W}}^* := \frac{\hat{\mathbf{W}} + \hat{\mathbf{W}}^T}{2}$$

2.4 Graph cuts

The basic idea is just to separate the vertices in different groups such that the edges within a group have high weight and the ones between different groups have low weight.

Definition 2.11. We define $\text{cut}(A_1, \dots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \sum_{v_j \in A_i, v_t \notin A_i} w_{jt}$.

The most basic way to formulate this is the minCut problem which corresponds to finding the sets that minimize the sum of the weights of edges between vertices of different sets (the sets that minimize the Cut function).

It has the problem that it tends to isolate a single point from the rest of the graph which gives way to non satisfactory clustering. To fix this, some solutions add a term that penalizes sets too small.

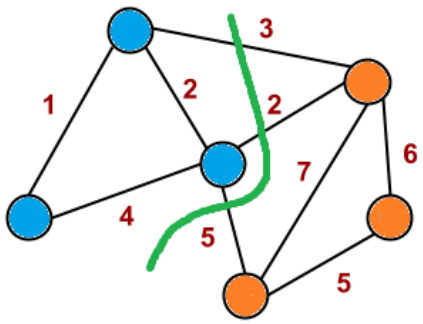

Figure 4: Example of a (non minimal) cut between two sets. The value of the cut is the sum of the weights of the edges crossed by the green line: 10 in this case.

Definition 2.12. The RatioCut corresponds to finding the sets $A_1, ..., A_k$ such that minimize:

$$\sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A_i})}{|A_i|}$$

Definition 2.13. We define the volume of a subset of vertices as:

$$\text{vol}(A) := \sum_{v_i \in A} d_i$$

where d_i is the degree of the vertex v_i.

Definition 2.14. The NCut corresponds to finding the sets $A_1, ..., A_k$ such that minimize:

$$\sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A_i})}{\text{vol}(A_i)}$$

This formulations enforce that the sets found are not too small by dividing by its cardinality or its total degree. Both this problems are NP-hard though, so it is not realistic to try to solve them for general graphs. But we can relax them into problems with closed form solution.

Relaxing RatioCut

Given a partition of V into k sets we define k indicator vectors $\mathbf{h}_j = (h_{1,j}, ..., h_{n,j})^T$ as

$$h_{i,j} = 1/\sqrt{|A_j|} \text{ if } v_i \in A_j, \text{ or 0 otherwise.}$$

We set $\mathbf{H} \in \mathbb{R}^{n \times k}$ as the matrix containing those k indicator vectors as columns. Observe that $\mathbf{H}^T\mathbf{H} = \mathbf{I}$. Also observe that:

$$\mathbf{h}_j^T \mathbf{L} \mathbf{h}_j = (\mathbf{H}^T \mathbf{L} \mathbf{H})_{jj} = \frac{\text{cut}(A_j, \overline{A_j})}{|A_j|}$$

Combining these we get that:

$$\text{RatioCut}(A_1, ..., A_k) = \text{Tr}(\mathbf{H}^T \mathbf{L} \mathbf{H})$$

We can relax the constraint of the structure of $\mathbf{H}$ and just ask that $\mathbf{H}^T\mathbf{H} = \mathbf{I}$.

$$\begin{aligned} \underset{\mathbf{H}}{\text{minimize}} \quad & \text{Tr}(\mathbf{H}^T\mathbf{L}\mathbf{H}) \\ \text{subject to} \quad & \mathbf{H}^T\mathbf{H} = \mathbf{I} \end{aligned}$$

The Rayleigh-Ritz theorem tells that the solution is given by $\mathbf{H}$ such that contains the k eigenvectors of $\mathbf{L}$ corresponding to the k smallest eigenvalues.

Relaxing NCut

Something similar can be done with NCut. We define the indicator vectors $\mathbf{h}_j = (h_{1,j}, ..., h_{n,j})^T$ by:

$$h_{i,j} = 1/\sqrt{\text{vol}(A_j)} \text{ if } v_i \in A_j, \text{ or 0 otherwise.}$$

We set $\mathbf{H} \in \mathbb{R}^{n \times k}$ as the matrix containing those k indicator vectors as columns. Observe that $\mathbf{H}^T\mathbf{D}\mathbf{H} = \mathbf{I}$. Also observe that:

$$\mathbf{h}_j^T\mathbf{L}\mathbf{h}_j = (\mathbf{H}^T\mathbf{L}\mathbf{H})_{jj} = \frac{\text{cut}(A_j, \overline{A_j})}{\text{vol}(A_j)}$$

We can relax the constraint of the structure of $\mathbf{H}$ and just ask that $\mathbf{H}^T\mathbf{D}\mathbf{H} = \mathbf{I}$.

$$\begin{aligned} \underset{\mathbf{H}}{\text{minimize}} \quad & \text{Tr}(\mathbf{H}^T\mathbf{L}\mathbf{H}) \\ \text{subject to} \quad & \mathbf{H}^T\mathbf{D}\mathbf{H} = \mathbf{I} \end{aligned}$$

We can also substitute $\mathbf{T} = \mathbf{D}^{1/2}\mathbf{H}$ and we obtain:

$$\begin{aligned} \underset{\mathbf{T}}{\text{minimize}} \quad & \text{Tr}(\mathbf{T}^T\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}\mathbf{T}) \\ \text{subject to} \quad & \mathbf{T}^T\mathbf{T} = \mathbf{I} \end{aligned}$$

which, in a similar fashion, has as solution the eigenvectors of the normalized symmetric Laplacian.

Remark 2.15. We should observe that after the relaxation, the solution that we obtain is not feasible for the original problem.

2.5 Algorithm

RatioCut - Unnormalized spectral clustering

Given a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ and the number k of clusters to construct:

1. Compute $\mathbf{L} = \mathbf{D} - \mathbf{W}$ (unnormalized Laplacian).
2. Compute the k smallest eigenvectors of $\mathbf{L}$ and let $\mathbf{U}$ be the matrix containing them as columns.
3. Cluster the n rows (let's call them $\mathbf{y}_j \in \mathbb{R}^k,\ j = 1 \dots n$) of the matrix $\mathbf{U}$ using the k-means algorithm.

The output are the clusters $A_1, ...A_k$ with $A_i = \{\mathbf{x}_j | y_j \in C_i\}$.
There are versions that come from the Ncut relaxation such as:

- Ng, Jordan and Weiss algorithm [14]: uses the eigenvectors of the $\mathbf{L}_{sym}$ and normalizes the rows of the matrix $\mathbf{U}$ before applying k-means.
- Shi and Malik [18]: uses the eigenvectors of the $\mathbf{L}_{rw}$ which correspond to the generalized eigenvectors of the problem.

2.6 Intuitive idea of spectral clustering

Let's remember the result we have presented before in Theorem 2.7 to give an intuitive idea of why the algorithm works.

Theorem 2.16 (Eigenspace of Laplacians)**.** *Each Laplacian has* 0 *as eigenvalue and its multiplicity k is the number of connected components* $A_1, ...A_k$ *of the graph. Also, the kernel of the unnormalized Laplacian is spanned by the indicator vectors* $\mathbf{1}_{A_1}, ..., \mathbf{1}_{A_k}$ *of those components.*

If we consider this theorem, we can establish a comparison between the ideal case and the real case:

- **Ideal case:** the graph has k connected components, one for each cluster. Thus the Laplacian has eigenvalue 0 with multiplicity k.
  The k eigenvectors with eigenvalue 0 are linear combinations of the indicator vectors which let us identify the clusters as they belong in the subspace spanned by the indicator vectors.
- **In practice:** small eigenvalues are an approximation of the ideal zero eigenvalues and eigenvectors are an approximation of the indicator vectors. If the approximation is good enough, we can recover the ideal connected components when doing the k-means step.

Connecting with this interpretation, there is an heuristic method to choose the number k of clusters to find. If $0 = \lambda_1 \leq ... \leq \lambda_n$ are the eigenvalues of $\mathbf{L}$ in increasing order, we can look at the eigengap: $\max_k \lambda_{k+1} - \lambda_k$. This eigengap can give us an intuition on which values are the approximation of the zero eigenvalues and which are not.

2.7 Comparing performance of clusterings

There are many different ways to measure if the clustering obtained is good, but we will restrict ourselves to comparing with some ground truth information.

Definition 2.17. A clustering C is a partition of a set of points or data set D into sets C_1, ... C_k with $|C_i| = n_i > 0$, $i = 1 ... k$ and $|D| = n$.

In this section we will consider that C' is the "perfect" clustering, the ground truth.

Purity

One of the most basic ones is to consider the purity of the clustering [10].

Definition 2.18. If C' is the "ideal" clustering of the data set, we can define the purity of our clustering as:

$$\text{purity}(C, C') = \frac{1}{n} \sum_k \max_j |C_k \cap C'_j|$$

Purity has a lower bound of 0 and an upper bound of 1 (higher means better clustering) and is very easy to interpret. It has the downfall that high purity is easy to achieve if the number of clusters is large (if every point gets its own cluster then we get maximum purity).

Variation of information

Definition 2.19. The mutual information between two clusterings is defined as:

$$I(C, C') = \sum_{k,k'} P(k, k') \log \frac{P(k, k')}{P(k)P(k')}$$

where $P(k, k')$ is the probability that a point belongs to C_k in clustering C and to C'_k in C'.

Intuitively it gives us the reduction of uncertainty that we have in C' when we are told where the points are clustered in the clustering C.

Some properties:

- It is symmetric and nonnegative
- $I(C, C') \leq \min\{H(C), H(C')\}$ having equality when one cluster completely determines the other

Definition 2.20. We define the variation of information [12]

$$VI(C, C') = H(C) + H(C') - 2I(C, C')$$

It satisfies that it is a distance function. Also it is lower bounded by $2 \log K$ where K is the number of clusters.

Normalized mutual information

Definition 2.21. We can define:

$$\text{NMI}(C, C') = \frac{I(C, C')}{(H(C) + H(C'))/2}$$

Now the index takes values between 0 and 1 (with 0 being that one clustering does not give any information about the other). This way we get an index that is easier to interpret. This is the measure we have used in all the experiments in this thesis.

2.8 Performance of different Laplacians

We want to test the performance of the different Laplacians associated to different spectral clustering algorithms. To experiment we have chosen different data sets: some from real life experiments, some that try to represent synthetic shapes and randomized weight matrix with noise.

Synthetic shapes

The synthetic shapes correspond to random data points in $\mathbb{R}^2$ that try to look like certain figure. Specifically we have used the figure of two moons and the figure of three circles. They are created using the "clusterSim" package in R. In the experiments we have considered the self tuning Gaussian graph with 7 neighbors.

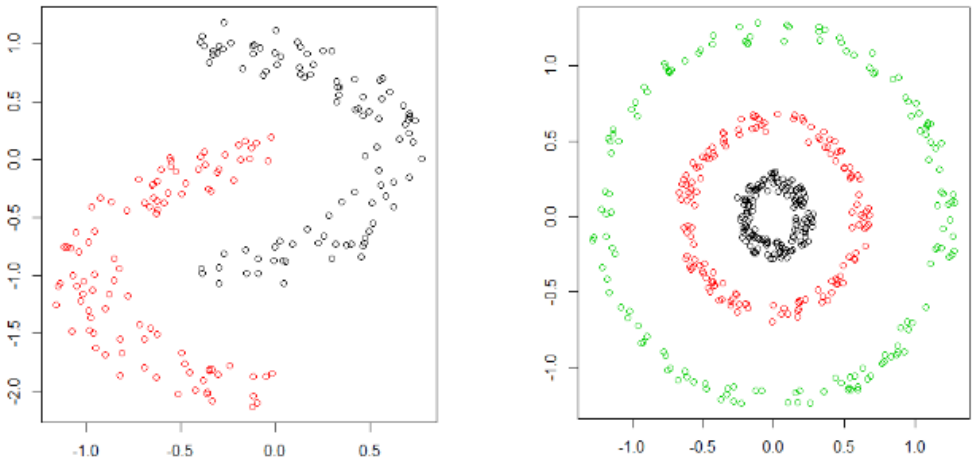

Figure 5: Plot of the two different kind of synthetic shapes used. Every cluster is in a different color.

Real life data

We have used the Yeast data set [2] which contains 1484 data points, 8 variables and data should belong to 9 different clusters. It tries to predict the localization site of a protein (non-numerically). Another data set used is the famous Iris data set [2]. It contains 150 points with 4 variables and the 3 different clusters correspond to different kind of flowers. In the experiments we have considered the self tuning Gaussian graph with 7 neighbors.

Noisy weight matrices

Definition 2.22. A noisy weight matrix is a symmetric matrix in $[0,1]^{n\times n}$ such that the elements that are in certain blocks along the diagonal (the clusters) belong to a $U(0,1)$ and the elements outside this blocks belong to a $U(0,c)$ for some $0 < c < 1$.

Each one represents a weight matrix with randomly weighted connections. To be useful for the experiments, inside the clusters the weights are higher on average. As we can see in the Figure 6 the "yellower" blocks along the diagonal correspond to the clusters. We have use different kinds of noisy affinity matrices:

- $n = 100$ with 4 "balanced" clusters of size 25 and varying values for c.
- $n = 300$ with 5 "unbalanced" clusters of size $20, 40, 40, 100, 100$ and varying values for c.
- $n = 100$ with 10 "small" clusters of size 10 and varying values for c.

We have repeated every clustering 100 times to try to reduce the effect of the randomness of the data (except for real life data sets, as they do not change every repetition).

In the following table 1 there is the average NMI for each Laplacian and data set. In red there is the maximum result for each data set. The Laplacians named "reg." are regularized normalized Laplacians with regularization constant equal to the next value.

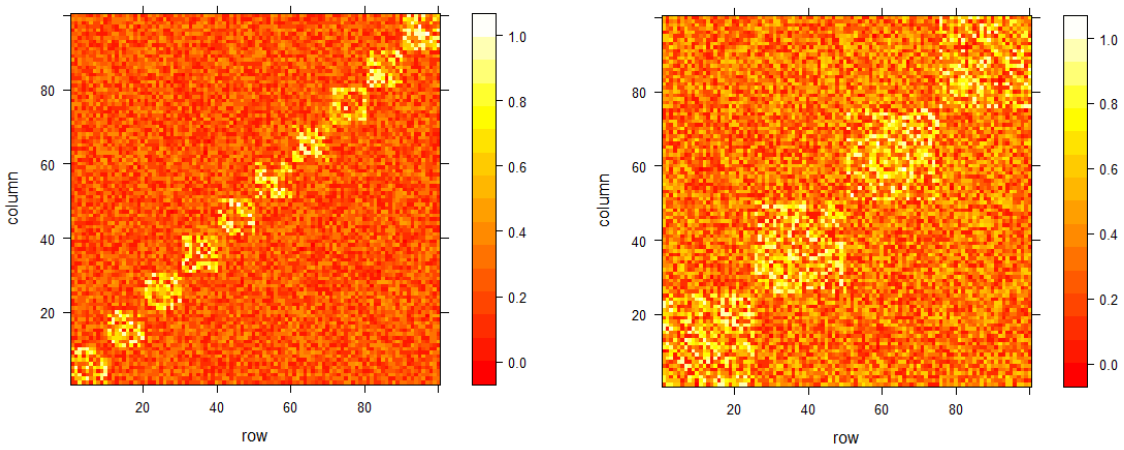

Figure 6: Level plot of two of the different weight matrices. The left image corresponds to a "small" clusters weight matrix with $c = 0.4$ and the right one to a "balanced" clusters with $c = 0.65$.

The result we get is that the unnormalized Laplacian works really worse in our tests. There is not a single case where we gain anything by choosing it instead of the normalized symmetric Laplacian or the random walk Laplacian. This may lead us to think that these are more useful for clustering and that solving the NCut problem is more helpful in real life than solving the RatioCut one.

	unnormalized	norm. sym.	r. walk	reg. 0.25	reg. 0.5	reg. 1
two moons	0.73	0.77	0.74	0.62	0.61	0.59
three circles	0.99	0.99	0.99	0.43	0.42	0.42
iris	0.77	0.79	0.77	0.82	0.82	0.82
yeast	0.23	0.25	0.27	0.26	0.26	0.26
balanced 0.65	0.49	0.91	0.90	0.91	0.91	0.91
balanced 0.75	0.14	0.44	0.44	0.44	0.45	0.45
balanced 0.85	0.06	0.12	0.12	0.12	0.12	0.12
unbalanced 0.6	0.66	0.92	0.94	0.88	0.87	0.87
unbalanced 0.7	0.22	0.80	0.79	0.79	0.79	0.79
unbalanced 0.8	0.07	0.43	0.43	0.42	0.42	0.42
small 0.4	0.78	0.99	0.97	0.99	0.99	0.98
small 0.5	0.49	0.87	0.86	0.87	0.87	0.87
small 0.6	0.31	0.59	0.59	0.59	0.60	0.59

Table 1: Values of NMI for the different Laplacians and data sets. In red there are the maximum results for each data set.

3. Graph learning for random variables

In the last section we saw some methods to obtain graph representations from general data sets. We only asked for a distance measure between different data points. But if our data consists of a collection of independent realizations of a random variable, we may consider other ways to get a similarity between data points.

3.1 Correlation graph

Definition 3.1. We define the correlation of the vector of random variables X as:

$$\mathbf{C} = (\text{diag}(\mathbf{\Sigma}))^{-1/2}\, \mathbf{\Sigma}\, (\text{diag}(\mathbf{\Sigma}))^{-1/2}$$

where $\mathbf{\Sigma}$ is the covariance matrix of X and $\text{diag}(\mathbf{\Sigma})$ is the diagonal matrix that has the same diagonal as $\mathbf{\Sigma}$.

The absolute value of the correlation [8] between two time series can be used as a measure of similarity and this way we can construct an adjacency matrix out of the correlation matrix.

We could use as a weight matrix:

$$\mathbf{W} := |\mathbf{C}| - \mathbf{I}$$

where $|\mathbf{C}|$ corresponds to the entrywise absolute value of the correlation matrix.

3.2 Conditional independence graphs

Conditional independence

Definition 3.2. We say that two random variables X and Y are conditionally independent given a third variable Z if their conditional probability distributions given Z are independent. We denote it by $X \perp Y|Z$.

Example 3.3. An intuitive example of the concept is the following: Height and vocabulary of kids are not independent, but they are conditionally independent if you also account for age.

Definition 3.4. We define precision matrix as $\mathbf{\Theta} := \mathbf{\Sigma}^{-1}$ where $\mathbf{\Sigma}$ is the covariance matrix.

If we consider that our random variables are Gaussian with zero mean and covariance $\mathbf{\Sigma}$, we have a nice result:

Theorem 3.5. *The graph $G = (V, E)$ with $V = \{v_1, v_2, ..., v_n\}$ and $E = \{v_i v_j \mid \mathbf{\Theta}_{ij} \neq 0, v_i, v_j \in V\}$ satisfies that*

$$v_i v_j \notin E \iff x_i \perp x_j \mid \{x_1, ..., x_n\} \setminus \{x_i, x_j\}$$

In other words, if we consider the graph that has edges corresponding to the non-zero entries of $\mathbf{\Theta}$ then this edges indicate conditional dependence between the two random variables given all the other random variables.

To prove it we need a previous proposition first:

Proposition 3.6 (Conditional Normal distribution)**.** *If* $Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \sim N(0, \mathbf{\Sigma})$ *and* $\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{bmatrix}$ *then* $Y_1 \mid Y_2 \sim N(0, \mathbf{\Sigma}_{11} - \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}\mathbf{\Sigma}_{21})$.

Proof. Using the proposition 3.6 we can check that $\left(\mathbf{\Sigma}_{11} - \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}\mathbf{\Sigma}_{21}\right)^{-1} = \left(\mathbf{\Sigma}^{-1}\right)_{11} = \mathbf{\Theta}_{11}$.

Now let's take $Y_1 = \begin{bmatrix} Y_a \\ Y_b \end{bmatrix}$ and Y_2 the rest. From the above results we know that the conditional random variable $Y_1 \mid Y_2$ follows a $N\left(0, \begin{bmatrix} \mathbf{\Theta}_{aa} & \mathbf{\Theta}_{ab} \\ \mathbf{\Theta}_{ba} & \mathbf{\Theta}_{bb} \end{bmatrix}^{-1}\right)$.

Trivially we can see now that if $\mathbf{\Theta}_{ab} = 0$ then since $\mathbf{\Theta}$ is symmetric $\mathbf{\Theta}_{ba} = 0$, the inverse is also diagonal and thus Y_a and Y_b are independent given Y_2. In the other way, if Y_a and Y_b are independent given Y_2 then $Y_1 \mid Y_2$ has a diagonal covariance matrix and thus $\mathbf{\Theta}_{ab} = 0$. □

Learning a sparse precision

Remark 3.7. So if we get a precision matrix we get a meaningful graph: two vertices will only be connected if the variables are conditionally dependent given all the others.

Also an important assumption we can make is that, in real life, interactions are mostly local and thus there are not many of them. So it would make sense to have a sparse precision matrix. In practice, though, we will only have access to a noisy sample covariance, so if we just invert it to get the precision we will not get zeros and we will lose this property. So we may want to estimate a sparse precision matrix and that can be done by solving the following convex optimization problem called graphical LASSO [7]:

$$\underset{\mathbf{\Theta} \succeq \mathbf{0}}{\text{maximize}} \;\; \log\det \mathbf{\Theta} - \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{\Theta}) - \rho \|\text{vec}(\mathbf{\Theta})\|_1$$

The two first terms come from the Maximum Likelihood estimation of the precision matrix under a Gaussian distribution and the last term with an l_1 norm induces sparsity on the matrix. This ρ parameter controls how much sparsity we want. Ideally we would have the cardinality of the matrix here to enforce sparsity, but that is not a convex function so that is why we use the l_1 norm. It can also be interpreted as a maximum a posteriori (MAP) estimation assuming a double exponential prior distribution.

3.3 Precision matrix with Laplacian structure

The precision matrices estimated with graphical LASSO do not give us a weighted graph directly since they dont have Laplacian or weight matrix structure in general (as they may have off diagonal elements with different signs). We might directly estimate a precision matrix with Laplacian structure [4] so we dont have to do any extra tricks.

$$\begin{aligned} \underset{\mathbf{\Theta} \succeq \mathbf{0}}{\text{minimize}} \quad & \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{\Theta}) - \log|\mathbf{\Theta}| + \rho\|\text{vec}(\mathbf{\Theta})\|_1 \\ \text{subject to} \quad & \mathbf{\Theta}_{ij} \leq 0 \;\; \forall i \neq j, \\ & \mathbf{\Theta}\mathbf{1} = \mathbf{0} \end{aligned}$$

The log determinant is substituted by the log pseudo determinant as now the matrix is singular but some mathematical tricks can be done in order to get a normal log determinant and an easier problem to solve.

Remark 3.8. The values of the inverse are related with the kind of partial correlation or covariance as it can be seen from the Proof 3.2. To learn a precision matrix with Laplacian structure implies learning a model where all the partial covariances are positive. So this may not be useful for general data sets.

To solve this problem we will use an iterative algorithm [22].

MM algorithm

If we assume that the graph we are looking for is a connected, we know that the Laplacian will have eigenvalue 0 with multiplicity 1 and eigenvector $\mathbf{1}$. Then:

$$\log|\mathbf{\Theta}| = \sum_{i=1}^{n-1}\log(\lambda_i) = \sum_{i=1}^{n-1}\log(\lambda_i) + \log(1) = \log\det(\mathbf{\Theta} + \mathbf{J}/n)$$

where $\mathbf{J} = \mathbf{1}\mathbf{1}^T$ and λ_i for $i = 1 \dots n-1$ are the positive eigenvalues of $\mathbf{\Theta}$. This way we get rid of the $\log|\mathbf{\Theta}|$.

Following, using that $\mathbf{\Theta}$ is a Laplacian, we can write it as a sum of weight contributions for each edge:

$$\mathbf{\Theta} = \sum_{i=1}^{n}\sum_{j>i} w_{ij}\left(\mathbf{e}_i\mathbf{e}_i^T - \mathbf{e}_i\mathbf{e}_j^T - \mathbf{e}_j\mathbf{e}_i^T + \mathbf{e}_j\mathbf{e}_j^T\right) =$$

$$\sum_{i=1}^{n}\sum_{j>i} w_{ij}\left(\mathbf{e}_i - \mathbf{e}_j\right)\left(\mathbf{e}_i - \mathbf{e}_j\right)^T =: \mathbf{E}\mathrm{Diag}(\mathbf{w})\mathbf{E}^T$$

where $\mathbf{E}$ is the incidence matrix, $\mathbf{w}$ is a vector that contains the weights of the edges and $\mathrm{Diag}(\mathbf{w})$ is the diagonal matrix that contains the vector $\mathbf{w}$ in the diagonal. This way the Laplacian constraints simplify to $\mathbf{w} \geq \mathbf{0}$.

Finally let's observe another thing:

$$\|\mathrm{vec}(\mathbf{\Theta})\|_1 = \mathrm{Tr}(\mathbf{\Theta}(2\mathbf{I} - \mathbf{J}))$$

so to simplify the formulation, let's define $\mathbf{K} := \hat{\mathbf{\Sigma}} + \rho(2\mathbf{I} - \mathbf{J})$. Also let's rewrite: $\mathbf{E}\mathrm{Diag}(\mathbf{w})\mathbf{E}^T + \mathbf{J}/n = [\mathbf{E}, \mathbf{1}]\mathrm{Diag}([\mathbf{w}, 1/n])[\mathbf{E}, \mathbf{1}]^T := \mathbf{G}\mathrm{Diag}([\mathbf{w}, 1/n])\mathbf{G}^T$.

The original formulation becomes now:

$$\underset{\mathbf{w}\geq\mathbf{0}}{\text{minimize}} \quad \mathrm{Tr}(\mathbf{E}\mathrm{Diag}(\mathbf{w})\mathbf{E}^T\mathbf{K}) - \log\det(\mathbf{G}\mathrm{Diag}([\mathbf{w}, 1/n])\mathbf{G}^T)$$

MM framework

The MM method [19], which stands for Majorization-Minimzation, can be used to solve general optimization problems of the form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{X} \end{aligned}$$

where f is differentiable. So instead of minimizing $f(x)$ we iteratively solve:

$$x^{(l+1)} \in \arg\min_{x\in\mathcal{X}} \bar{f}\left(x; x^{(l)}\right)$$

where $\bar{f}\left(x; x^{(l)}\right)$ should satisfy:

1. $\bar{f}\left(x; x^{(l)}\right) \geq f(x), \forall x \in \mathcal{X}$;
2. $\bar{f}\left(x^{(l)}; x^{(l)}\right) = f(x^{(l)})$;
3. $\nabla\bar{f}\left(x^{(l)}; x^{(l)}\right) = \nabla f(x^{(l)})$;
4. $\bar{f}\left(x; x^{(l)}\right)$ is continuous in both x and $x^{(l)}$.

We can apply this framework to our formulation and get an iterative problem with a closed form solution:

1. Using the concavity of the log-determinant function:

$$-\log\det(\mathbf{G}\mathbf{X}\mathbf{G}^T) \leq \mathrm{Tr}\left(\mathbf{F}_0(\mathbf{G}\mathbf{X}\mathbf{G}^T)^{-1}\right) + \text{const.}$$

   with $\mathbf{F}_0 := \mathbf{G}\mathbf{X}_0\mathbf{G}^T$ and $\mathbf{X} := \mathrm{Diag}([\mathbf{w}, 1/n])$.

2. We also have:

$$\mathrm{Tr}\left(\mathbf{F}_0(\mathbf{G}\mathrm{Diag}([\mathbf{w}, 1/n])\mathbf{G}^T)^{-1}\right) = \mathrm{Tr}\left(\mathbf{F}_0^{1/2}(\mathbf{G}\mathrm{Diag}([\mathbf{w}, 1/n])\mathbf{G}^T)^{-1}\mathbf{F}_0^{1/2}\right) \leq$$

$$\mathrm{Tr}\left(\mathbf{F}_0^{-1/2}\mathbf{G}\mathrm{Diag}([\mathbf{w}_0, 1/n])(\mathrm{Diag}([\mathbf{w}, 1/n]))^{-1}\mathrm{Diag}([\mathbf{w}_0, 1/n])\mathbf{G}^T\mathbf{F}_0^{-1/2}\right)$$

and this functions satisfy the conditions necessary to be useful for MM.

Now we can rewrite the MM iterative problem as:

$$\underset{\mathbf{w}\geq\mathbf{0}}{\text{minimize}} \quad \mathrm{diag}(\mathbf{R})^T\mathbf{w} + \mathrm{diag}(\mathbf{Q})^T\mathbf{w}^{-1}$$

where $\mathbf{R} := \mathbf{E}^T\mathbf{K}\mathbf{E}$ and $\mathbf{Q} := \mathrm{Diag}([\mathbf{w}_0, 1/n])\mathbf{G}^T\left(\mathbf{G}\mathrm{Diag}([\mathbf{w}_0, 1/n])\mathbf{G}^T\right)^{-1}\mathbf{G}\mathrm{Diag}([\mathbf{w}_0, 1/n])$.

This problem has a closed form solution given by:

$$\mathbf{w}^* = \sqrt{\mathrm{diag}(\mathbf{Q}) \odot \mathrm{diag}(\mathbf{R})^{-1}}$$

where $\odot$ corresponds to the Hadamard (entrywise) matrix product and the square root is also entrywise.

3.4 Graph signal processing

Definition 3.9. The shift operator $\mathbf{S}$ of a graph is a linear operator that replaces a signal value at each vertex with the linear combination of the values at the neighbors of that vertex.

Remark 3.10. It can be interpreted as the graph equivalent of the time shift or delay z^{-1}. We can consider that the weight matrix or the Laplacian matrix of a graph are examples of symmetric shift operators.

One supposition [17] may be that the signal $\mathbf{x}$ comes from a filter on the symmetric shift operator $\mathbf{S}$ of the graph applied to white noise $\mathbf{w}$: $\mathbf{x} = \sum_{l\geq 0}\beta_l\mathbf{S}^l\mathbf{w}$. The signal processing equivalent would be to say that the signal comes from a FIR filter applied to white noise.

Proposition 3.11. *The shift operator* $\mathbf{S}$ *and the covariance matrix* $\mathbf{C}_\mathbf{x}$ *have a common base of eigenvectors:* $\mathbf{S}\mathbf{C}_\mathbf{x} = \mathbf{C}_\mathbf{x}\mathbf{S}$.

Proof. If $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ with $\mathbf{\Lambda}$ diagonal, then the filter $\mathbf{H}$ satisfies $\mathbf{H} := \sum_{l\geq 0} \beta_l \mathbf{S}^l = \mathbf{V}\left(\sum_{l\geq 0} \beta_l \mathbf{\Lambda}^l\right)\mathbf{V}^T$. Thus the covariance matrix $\mathbf{C_x} = \mathbb{E}(\mathbf{x}\mathbf{x}^T) = \mathbb{E}(\mathbf{H}\mathbf{w}(\mathbf{H}\mathbf{w})^T) = \mathbf{H}\mathbb{E}(\mathbf{w}\mathbf{w}^T)\mathbf{H}^T = \mathbf{H}\mathbf{H}^T$. In particular $\mathbf{C_x}$ has the same eigenvectors as $\mathbf{S}$. □

But knowing the eigenvectors is not enough. Thus we decide to use convex optimization to find a shift operator that shares eigenvectors with the covariance matrix but also has certain convex structure and is the most sparse by using the l_1 norm.

$$\begin{aligned} \underset{\mathbf{S},\lambda}{\text{minimize}} \quad & \|\text{vec}(\mathbf{S})\|_1 \\ \text{subject to} \quad & \mathbf{S} = \mathbf{V}\text{diag}(\lambda)\mathbf{V}^T \\ & \mathbf{S} \in \mathcal{S} \end{aligned}$$

where $\mathbf{V}$ is the matrix of eigenvectors of the covariance matrix $\mathbf{\Sigma}$. An example of convex structure might be that the matrix is a unnormalized Laplacian or an affinity matrix.

$$\begin{aligned} \mathbf{S} \in \mathcal{S}_A &\iff \mathbf{S}_{ij} \geq 0, \mathbf{S}_{ii} = 0, \mathbf{S} = \mathbf{S}^T, \forall i \neq j \\ \mathbf{S} \in \mathcal{S}_L &\iff \mathbf{S}_{ij} \leq 0, \mathbf{S}\mathbf{1} = 0, \mathbf{S} = \mathbf{S}^T, \forall i \neq j \end{aligned}$$

We should also add some scale constraint to avoid the null solution: for example $\sum_{i=2}^{n} |\mathbf{S}_{1i}| = 1$

Like before, we will not have normally the exact real covariance matrix as we only have the noisy sample covariance matrix $\hat{\mathbf{\Sigma}}$. So we should allow for some error in the eigenvectors. We can get that by adding an auxiliary term that shares the eigenvectors with the sample covariance and asking that this term and the shift operator are close in Frobenius norm.

$$\begin{aligned} \underset{\mathbf{S},\lambda,\mathbf{S}'}{\text{minimize}} \quad & \|\text{vec}(\mathbf{S})\|_1 + \frac{\alpha}{2}\|\mathbf{S} - \mathbf{S}'\|_F^2 \\ \text{subject to} \quad & \mathbf{S}' = \hat{\mathbf{V}}\text{diag}(\lambda)\hat{\mathbf{V}}^T \\ & \mathbf{S} \in \mathcal{S} \end{aligned}$$

where $\hat{\mathbf{V}}$ is the matrix of eigenvectors of the sample covariance matrix.

We include a reformulation of this problem as a quadratic programming problem (QP). Using this formulation, we can use specialized algorithms to find the solution.

QP formulation

For the case where the graph operator is an affinity matrix the relaxed weight formulation can be written as:

$$\begin{aligned} \underset{\mathbf{S},\mathbf{S}'}{\text{minimize}} \quad & \|\text{vec}(\mathbf{S})\|_1 + \frac{\alpha}{2}\|\mathbf{S} - \mathbf{S}'\|_F^2 \\ \text{subject to} \quad & \mathbf{S}'\mathbf{\Sigma} = \mathbf{\Sigma}\mathbf{S}' \\ & \mathbf{S}_{ij} = \mathbf{S}_{ji} \geq 0 \\ & \mathbf{S}_{ii} = 0 \\ & \sum_k \mathbf{S}_{1k} = 1 \end{aligned}$$

since $\mathbf{S}'$ and $\mathbf{\Sigma}$ commute if and only if they have a common base of eigenvectors.

The general formulation of a QP problem is:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{x}^T\mathbf{D}\mathbf{x} - \mathbf{d}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}^T\mathbf{x} \geq \mathbf{b} \end{aligned}$$

We will need to rewrite our problem in vector form: we will use as $\mathbf{x} \in \mathbb{R}^{n(n+1)}$ the following vector

$$\mathbf{x} = (\mathbf{S}_{11}, \mathbf{S}_{21}, \mathbf{S}_{22}, \mathbf{S}_{31}, \mathbf{S}_{32}, \mathbf{S}_{33}, \dots, \mathbf{S}_{nn}, \mathbf{S}'_{11}, \mathbf{S}'_{21}, \mathbf{S}'_{22}, \dots)^T$$

using the fact that both $\mathbf{S}$ and $\mathbf{S}'$ are symmetric (so we can take out of the formulation $n(n-1)$ variables).

Let's start by rewriting the objective function:

$$-\mathbf{d}^T\mathbf{x} = \|\text{vec}(\mathbf{S})\|_1$$

but using that $\mathbf{S}_{ij} \geq 0$ and $\mathbf{S}_{ii} = 0$ we can directly write:

$$\mathbf{d} := (-2, \dots, -2, 0, \dots, 0)^T$$

The other part of the objective function is a bit more tricky:

$$\frac{1}{2}\mathbf{x}^T\mathbf{D}\mathbf{x} = \frac{\alpha}{2}\|\mathbf{S} - \mathbf{S}'\|_F^2$$

Define $\mathbf{t}$ as the following vector:

$$\mathbf{t} := (1, \sqrt{2}, 1, \sqrt{2}, \sqrt{2}, 1, \dots)^T \in \mathbb{R}^{n(n+1)/2}$$

such that 1 corresponds to a diagonal element of $\mathbf{S}$ in $\mathbf{x}$ and $\sqrt{2}$ corresponds to a non diagonal element of $\mathbf{S}$ in $\mathbf{x}$.

Now define $\mathbf{T}$ as the diagonal matrix that contains the vector $\mathbf{t}$ in the diagonal:

$$\mathbf{T} := \text{Diag}(\mathbf{t})$$

Now we can simply check that:

$$\mathbf{D} := \alpha \begin{bmatrix} \mathbf{T} & -\mathbf{T} \\ -\mathbf{T} & \mathbf{T} \end{bmatrix}$$

satisfies the desired equality.

The only constraint that is a bit tricky to write is $\mathbf{\Sigma}\mathbf{S}' = \mathbf{S}'\mathbf{\Sigma}$. It is equivalent to saying:

$$\mathbf{\Sigma}_i \cdot \mathbf{S}_j'^T = \mathbf{\Sigma}_j \cdot \mathbf{S}_i'^T, \ \forall i, j = 1 \dots n$$

where the sub index i refers to the i-th row of the matrix.

This equality is trivial for $i = j$ so we just have to consider all the other $n(n-1)/2$ cases. For the other cases we will choose $i < j$ without loss of generality, we only need to consider the row of the matrix $\mathbf{A}$ as a vector with:

1. 0 in the positions corresponding to $\mathbf{S}$: the first $n(n+1)/2$.
2. $\mathbf{\Sigma}_{ik}$ in the position corresponding to $\mathbf{S}'_{jk}$ or $\mathbf{S}'_{kj}$ for $k \neq i$.
3. $-\mathbf{\Sigma}_{jk}$ in the position corresponding to $\mathbf{S}'_{ik}$ or $\mathbf{S}'_{ki}$ for $k \neq j$.
4. 0 in the position corresponding to $\mathbf{S}'_{ji}$.

4. Modelling stock data

We consider the daily log-returns of stock data [5] because of their statistical properties: they fit better than other quantities into a multivariate Gaussian, although not perfectly, as they have heavier tails.

To simplify the study we assume a basic i.i.d. Gaussian model with one explicit factor:

$$\mathbf{x} = \alpha + f_M \cdot \beta + \epsilon$$

where f_M is the random variable corresponding to the market factor, α is the expected mean, β is the factor loading vector and ϵ are the residuals. Thus $x \sim N(\alpha, \mathbf{\Sigma})$ with $\mathbf{\Sigma} = \beta\beta^T + \text{cov}(\epsilon)$.

The reason for including this global factor explicitly is because stocks are very correlated with the whole market [8]. It has a large effect on the covariance matrix and thus, maybe, we should study it separately.

4.1 Effect of a global factor

To study if $\mathbf{\Sigma}$ and $\text{cov}(\epsilon)$ have the same conditional independence pattern, we introduce:

Theorem 4.1 (Sherman-Morrison inversion formula)**.** *If* $\mathbf{A}$ *is an invertible matrix,* $\mathbf{u}$ *is a column vector and* $\mathbf{A} + \mathbf{u}\mathbf{u}^T$ *is also invertible then:*

$$\left(\mathbf{A} + \mathbf{u}\mathbf{u}^T\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{A}^{-1}}{1 + \mathbf{u}^T\mathbf{A}^{-1}\mathbf{u}}.$$

Proof. Trivial, just multiply to see that it is the inverse. □

Let's check the effect of a global factor on the sparsity pattern of the inverse: if we put $\mathbf{u} = \beta = \mathbf{1}$ for example, and $\mathbf{A} = \text{cov}(\epsilon) =: \mathbf{\Psi}$ (with $\mathbf{\Psi} + \beta\beta^T = \mathbf{\Sigma}$). Thus:

$$\mathbf{\Sigma}^{-1} = \mathbf{\Psi}^{-1} - \frac{\mathbf{\Psi}^{-1}\mathbf{1}\mathbf{1}^T\mathbf{\Psi}^{-1}}{1 + \mathbf{1}^T\mathbf{\Psi}^{-1}\mathbf{1}}$$

and in general we can see that $\mathbf{\Sigma}^{-1}$ and $\mathbf{\Psi}^{-1}$ have different sparsity patterns (they do not have 0 in the same positions). This way if we estimate the graph with $\mathbf{\Sigma}^{-1}$ or $\mathbf{\Psi}^{-1}$ we get different outcome. Moreover, we may get better results if we use $\mathbf{\Psi}^{-1}$.

PCA estimation

One way to estimate β is by setting it equal to the eigenvector of the covariance matrix with bigger eigenvalue and the variance of the factor equal to the corresponding eigenvalue. The idea behind is that since the market factor is much more influential than any other effect, then the biggest eigenvalue corresponds mostly to this market factor.

ML estimation

Another popular statistical factor model estimation [9] is based on maximum likelihood estimation (ML) . Here we assume that the covariance matrix without the factor is diagonal and that the returns are Gaussian.

The problem to be solved is the following with $K = 1$:

$$\begin{aligned} \underset{\mathbf{B},\mathbf{\Psi}}{\text{minimize}} \quad & -\log|\mathbf{\Sigma}^{-1}| + \text{Tr}\left(\mathbf{\Sigma}^{-1}\hat{\mathbf{S}}\right) \\ \text{subject to} \quad & \mathbf{\Sigma} = \mathbf{B}\mathbf{B}^T + \mathbf{\Psi} \\ & \mathbf{B} \in \mathbb{R}^{N\times K} \\ & \mathbf{\Psi} = \text{diag}\,(\psi_1, \dots, \psi_p) \geq \epsilon\mathbf{I} \end{aligned}$$

Extended graph

Once we have estimated the factor β either way, we can directly get $\mathbf{\Psi} := \text{cov}(\epsilon) = \mathbf{\Sigma} - \beta\beta^T$. But if we are estimating a conditional independence graph, we have another option: we can also add an extra point that behaves as the global market factor. In that case, the new covariance matrix would be:

$$\begin{bmatrix} \mathbf{\Psi} + \beta\beta^T & \beta \\ \beta^T & 1 \end{bmatrix}$$

Using the matrix inversion lemma:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & -(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} \end{bmatrix}$$

we get that

$$\begin{bmatrix} \mathbf{\Psi} + \beta\beta^T & \beta \\ \beta^T & 1 \end{bmatrix}^{-1} = \begin{bmatrix} (\mathbf{\Psi}+\beta\beta^T-\beta\beta^T)^{-1} & -(\mathbf{\Psi}+\beta\beta^T-\beta\beta^T)^{-1}\beta 1^{-1} \\ -1^{-1}\beta^T(\mathbf{\Psi}+\beta\beta^T-\beta\beta^T)^{-1} & 1^{-1}+1^{-1}\beta^T(\mathbf{\Psi}+\beta\beta^T-\beta\beta^T)^{-1}\beta 1^{-1} \end{bmatrix} =$$

$$= \begin{bmatrix} (\mathbf{\Psi})^{-1} & -(\mathbf{\Psi})^{-1}\beta 1^{-1} \\ -1^{-1}\beta^T(\mathbf{\Psi})^{-1} & 1^{-1}+1^{-1}\beta^T(\mathbf{\Psi})^{-1}\beta 1^{-1} \end{bmatrix} = \begin{bmatrix} \mathbf{\Psi}^{-1} & -\mathbf{\Psi}^{-1}\beta \\ -\beta^T\mathbf{\Psi}^{-1} & 1+\beta^T\mathbf{\Psi}^{-1}\beta \end{bmatrix}$$

And thus the effect of the factor disappears of the conditional independence graph as we get $\mathbf{\Psi}^{-1}$ in the upper left block.

4.2 Latent variable model

We may also use convex optimization to jointly learn $\mathbf{\Sigma}$ and a factor β (or more than one) such that $\mathbf{\Sigma} - \beta\beta^T$ is sparse using a formulation similar to graphical LASSO. To do this we will use what we saw in Section 4.1: the effect of a factor can be interpreted as the effect of an unobserved variable.

Assume we have $x = \begin{bmatrix} x_O & x_L \end{bmatrix}^T$ a Gaussian random vector where x_O are the observed variables and x_L are the latent ones (in our case just one). Thus $x \sim N(0, \begin{bmatrix} \mathbf{\Sigma}_{O,O} & \mathbf{\Sigma}_{O,L} \\ \mathbf{\Sigma}_{L,O} & \mathbf{\Sigma}_{L,L} \end{bmatrix})$ and let $\mathbf{\Theta}$ be the inverse of this covariance matrix.

If we take the marginal without the latent variables (what we can observe) then we get a observed covariance $\mathbf{\Sigma}_{OO}$ and a observed precision $\tilde{\mathbf{\Theta}} = \mathbf{\Sigma}_{O,O}^{-1} = \mathbf{\Theta}_{O,O} - \mathbf{\Theta}_{O,L}\mathbf{\Theta}_{L,L}^{-1}\mathbf{\Theta}_{L,O}$. Define $\mathbf{S} := \mathbf{\Theta}_{O,O}$ and $\mathbf{L} := \mathbf{\Theta}_{O,L}\mathbf{\Theta}_{L,L}^{-1}\mathbf{\Theta}_{L,O}$. Observe thus that $\tilde{\mathbf{\Theta}} = \mathbf{S} - \mathbf{L}$ and $\mathbf{S} = \mathbf{\Theta}_{O,O}$ is the desired conditional precision matrix.

Nuclear norm approximation

We want to find a conditional precision matrix ($\mathbf{S}$) such that it is sparse. In the other hand, since $r << p$ we know that $\mathbf{L}$ will be low rank (in the case of the stock market the rank of $\mathbf{L}$ would be 1). We can try to enforce the low-rankedness of $\mathbf{L}$ by using the nuclear norm.

Definition 4.2. The nuclear norm $\|\mathbf{X}\|_*$ of a matrix corresponds to the sum of its singular values.

Proposition 4.3. *If* $\mathbf{X} \succeq \mathbf{0}$ *then* $\|\mathbf{X}\|_* = Tr(\mathbf{X})$.

Proof.

$$\|\mathbf{X}\|_* = \sum_{i=1}^{n} \rho_i(\mathbf{X}) = \sum_{i=1}^{n} \sqrt{\lambda_i(\mathbf{X}^T\mathbf{X})}$$

Since $\mathbf{X}$ is symmetric: $\mathbf{X}^T\mathbf{X} = \mathbf{X}^2$ and also using that the eigenvalues of $\mathbf{X}$ are bigger or equal than zero:

$$\sum_{i=1}^{n} \sqrt{\lambda_i(\mathbf{X}^T\mathbf{X})} = \sum_{i=1}^{n} \sqrt{\lambda_i(\mathbf{X}^2)} = \sum_{i=1}^{n} \sqrt{\lambda_i(\mathbf{X})^2} = \sum_{i=1}^{n} \lambda_i(\mathbf{X}) = \mathrm{Tr}(\mathbf{X})$$

□

The formulation [13] is:

$$\begin{aligned} \underset{\mathbf{S},\mathbf{L}}{\text{minimize}} \quad & \mathrm{Tr}\left(\hat{\mathbf{\Sigma}}(\mathbf{S}-\mathbf{L})\right) - \log\det(\mathbf{S}-\mathbf{L}) + \alpha\|\mathrm{vec}(\mathbf{S})\|_1 + \gamma\|\mathbf{L}\|_* \\ \text{subject to} \quad & \mathbf{L} \succeq \mathbf{0},\ \mathbf{S}-\mathbf{L} \succeq \mathbf{0} \end{aligned}$$

with the parameter γ to control the low rank of $\mathbf{L}$ and we can substitute $\|\mathbf{L}\|_* = \mathrm{Tr}(\mathbf{L})$. Ideally we would use the constraint $\mathrm{rank}(\mathbf{L}) \leq 1$, but it is a nonconvex constraint.

Fixed rank approximation

We might also fix the rank of $\mathbf{L}$ to $k < n$ by setting $\mathbf{L} = \mathbf{B}\mathbf{B}^T$ with $\mathbf{B} \in \mathbb{R}^{n\times k}$ (in the case of the stock market $k = 1$). The formulation is:

$$\begin{aligned} \underset{\mathbf{S},\mathbf{B}}{\text{minimize}} \quad & \mathrm{Tr}\left(\hat{\mathbf{\Sigma}}(\mathbf{S}-\mathbf{B}\mathbf{B}^T)\right) - \log\det(\mathbf{S}-\mathbf{B}\mathbf{B}^T) + \alpha\|\mathbf{S}\|_1 \\ \text{subject to} \quad & \mathbf{S}-\mathbf{B}\mathbf{B}^T \succeq \mathbf{0} \end{aligned}$$

But it is a nonconvex problem. We can reformulate the problem though:

$$\mathbf{S}-\mathbf{B}\mathbf{B}^T \succeq \mathbf{0} \iff \mathbf{X} = \begin{bmatrix} \mathbf{S} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{I} \end{bmatrix} \succeq \mathbf{0}$$

Proposition 4.4. *det*$(\mathbf{X}) =$ *det*$(\mathbf{S}-\mathbf{B}\mathbf{B}^T)$. *Specifically log det*$(\mathbf{S}-\mathbf{L}) =$ *log det*$(\mathbf{X})$.

Proof. We have the following equality:

$$\mathbf{X} = \begin{bmatrix} \mathbf{S} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{S}-\mathbf{B}\mathbf{B}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B}^T & \mathbf{I} \end{bmatrix}$$

□

This problem still has a non-convex objective function because of the trace part with $\mathbf{B}\mathbf{B}^T$. But we can use the MM framework presented before in Section 3.3 to get an iterative solution where every problem is convex. Since $\hat{\mathbf{\Sigma}} \succeq 0$ we can observe that:

$$0 \leq \text{Tr}(\hat{\mathbf{\Sigma}}(\mathbf{B}-\mathbf{B}_0)(\mathbf{B}-\mathbf{B}_0)^T) = \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}\mathbf{B}^T) - \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}_0\mathbf{B}^T) - \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}\mathbf{B}_0^T) + \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}_0\mathbf{B}_0^T),\ \forall \mathbf{B}, \mathbf{B}_0$$

and thus:

$$-\text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}\mathbf{B}^T) \leq -\text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}_0\mathbf{B}^T) - \text{Tr}(\hat{\mathbf{\Sigma}}\mathbf{B}\mathbf{B}_0^T) + \text{const.},\ \forall \mathbf{B}, \mathbf{B}_0$$

and moreover it satisfies the necessary conditions to be useful for MM.

So the problem has a convex iterative solution:

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \text{Tr}\left(\mathbf{X}\begin{bmatrix} \hat{\mathbf{\Sigma}} & -\hat{\mathbf{\Sigma}}\mathbf{B}_0 \\ -\mathbf{B}_0^T\hat{\mathbf{\Sigma}} & \mathbf{0} \end{bmatrix}\right) - \log\det(\mathbf{X}) + \alpha \left\| \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{X} \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \right\|_1 \\ \text{subject to} \quad & \mathbf{X} \succeq \mathbf{0},\ \mathbf{X}_{n+1:n+k,n+1:n+k} = \mathbf{I} \end{aligned}$$

starting with some initial $\mathbf{B}_0 \neq \mathbf{0}$ and after every iteration setting the $\mathbf{B}_0$ equal to the optimal $\mathbf{B}$ obtained. The last constraint $\mathbf{X}_{n+1:n+k,n+1:n+k} = \mathbf{I}$ corresponds to the condition that the last block is the $\mathbf{I}$.

Remark 4.5. If the initial $\mathbf{B}_0 = \mathbf{0}$, then $\mathbf{B}^* = \mathbf{0}$ and is an undesirable fixed point of the algorithm. The problem becomes equivalent to graphical LASSO in this case.

Remark 4.6. Since the initial problem is non-convex, we cannnot guarantee the algorithm converges to the global minimum of the problem, but we can guarantee it converges to a local minimum.

5. Experiments with stock data

The experiments with stock data have been done with stocks from SP500 from the sectors:

1. Real Estate
2. Consumer Staples
3. Energy

Ideally we would have used all the companies of SP500 but some of the algorithms were too slow for a data set so big.

Remark 5.1. The reason we have chosen this subset of stocks is because they clearly belong to very different sectors and thus it should be "easy" to cluster them. In other market sectors there might be more overlapping.

As data we will use daily log-returns for their statistical properties as we have discussed before. The exact list of companies used is present in the annex C.

5.1 Drawing algorithms for graphs

To draw the graphs we have used a force-directed algorithm. The algorithm considers the vertices as charged electrons and the edges as springs and tries to find a stable position (local minimum of energy) for them. These algorithms give intuitive drawings of the graphs though sometimes they get stuck in undesirable local minima. For this reason, the drawings obtained are not flawless.

In each graph, every vertex represents a stock and its color the sector it belongs to: in green the energy companies, in blue the real estate companies and in red the consumer staples ones. Every edge represents a relationship between stocks and its strength is proportional to its width. Finally, the color of an edge is determined by the color of the stocks it connects. The edges that connect stocks from different sectors are colored gray, representing its undesirability.

5.2 Cluster structure of stocks

We would like to learn graphs that represent the stocks structure and also separate stocks into clusters. This might be useful, for example, creating diversified portfolios less susceptible to market downturns.

As data we will use the daily log-returns from year 2018 from all the companies of the mentioned sectors.

Remark 5.2. It is important to observe that although we have many years of data it is not a good idea to take too much data. This is because we are approximating the market with certain parameters (specifically certain covariance matrix) but it is not stationary. Its better to use data on a rolling window basis.

All the methods use a parameter that affects the graph estimated. For example the regularization constant ρ of graphical LASSO. We have even added a parameter to the parameterless method: instead of considering the sample correlation matrix, we threshold it with some value, the threshold parameter. The possible values we have considered for the parameters are arbitrary, but we have tried to explore most of the parameter's space (for example for thresholding the correlation weight matrix we have considered the

values: $0, 0.05, 0.1, 0.15, \ldots, 0.95$ as the values are lower bounded by 0 and upper bounded by 1). Finally, to compare the performance of different parameters we have used spectral clustering on the graph obtained for each value (using the normalized symmetric Laplacian with a very small regularization constant). The parameters chosen are the ones that give higher NMI when comparing the graph obtained to the industry classification of stocks.

Some of the methods of learning (nuclear norm approximation, learning a shift operator with Laplacian constraints, ...) give non-exact results: the graphs obtained are only an approximation of the solution because we have used general purpose convex solvers. In those cases, we have "smoothed" the graph by thresholding it by a small value. Finally, the fixed rank latent model estimation could not be used because the algorithm was too slow for a data set of this size.

Remark 5.3. In the data set used in this experiment there is not any significant difference between the PCA and the ML estimation of the global factor. For this reason we will only use the PCA method.

Results

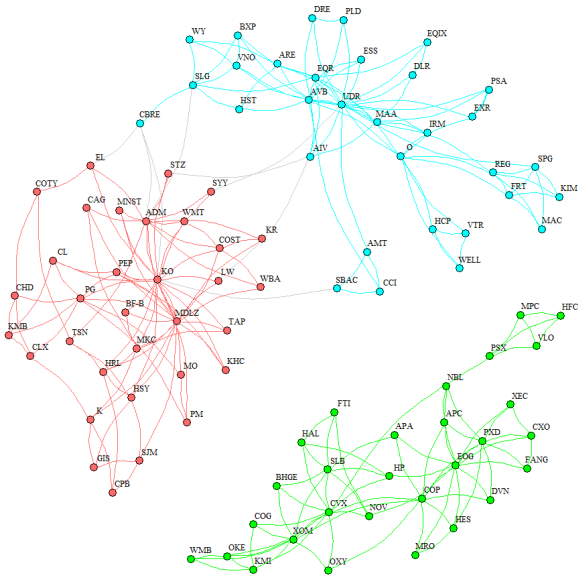

Figure 7: Jordan estimation graph with 3 neighbors. The NMI value is 0.96.

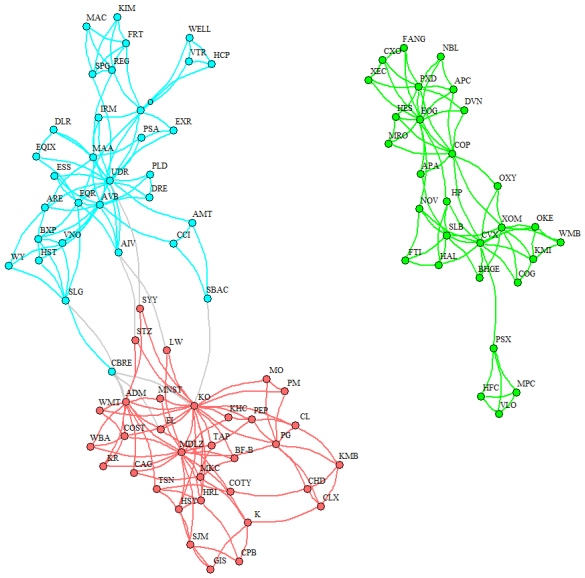

Figure 8: k-nearest neighbors graph with 3 neighbors. The NMI value is 0.96.

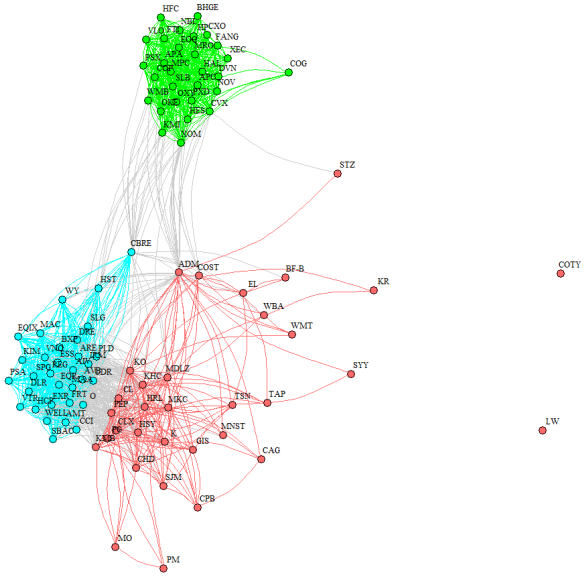

Figure 9: Correlation graph with 0.45 thresholding. The NMI value is 0.96.

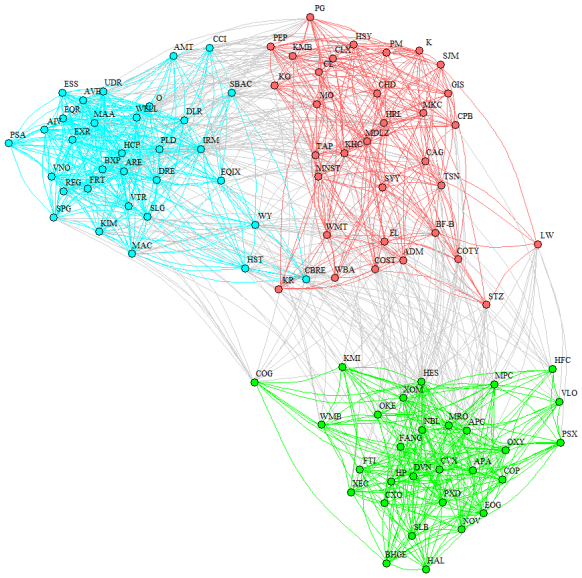

Figure 10: Graphical LASSO graph with regularization $\rho = e^{(} - 10)$. The NMI value is 0.96.

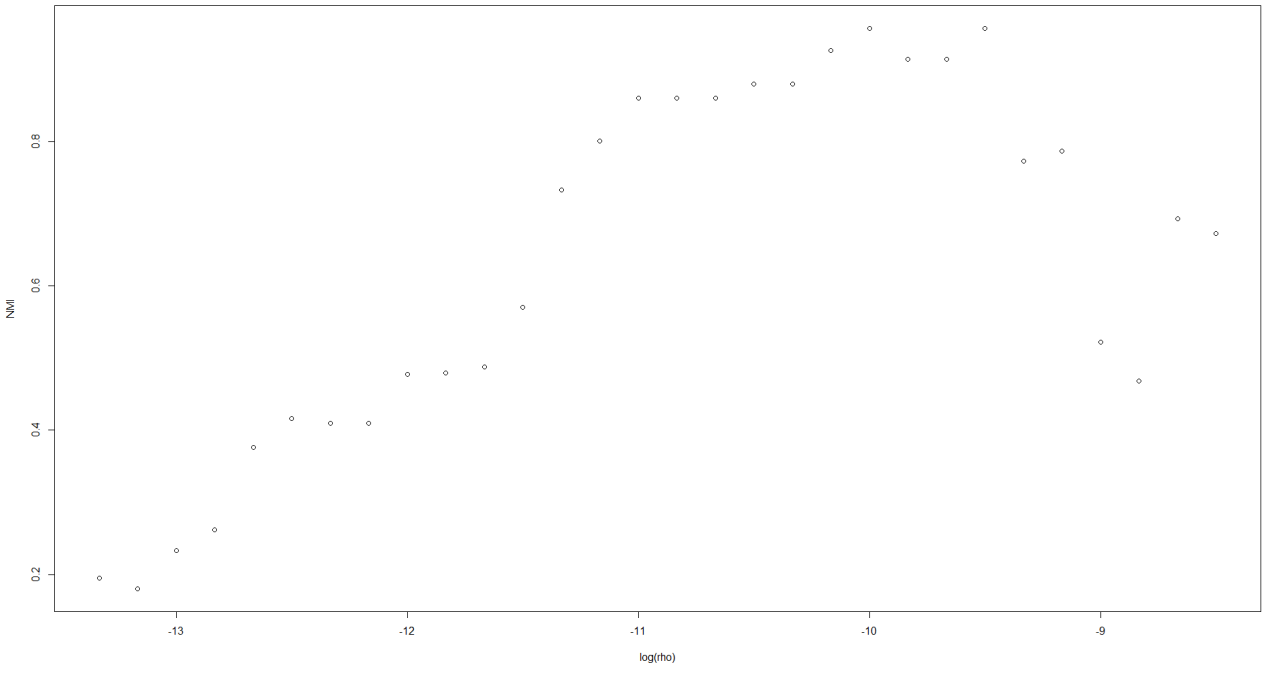

Figure 11: Plot of the NMI values in graphical LASSO estimation as a function of $\log(\rho)$

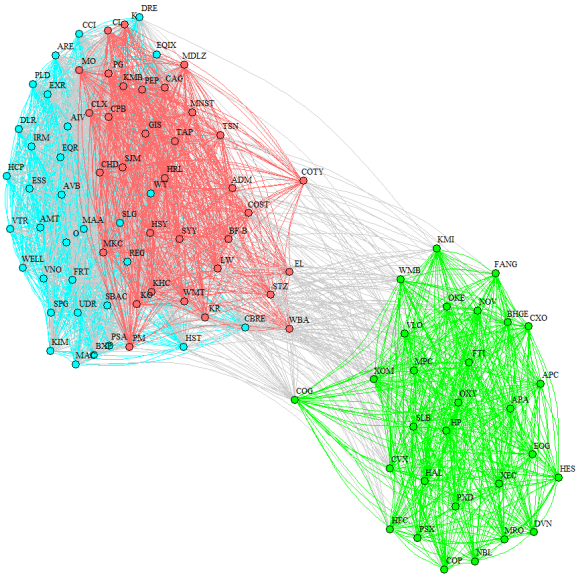

Figure 12: Laplacian precision graph with regularization $\rho = 3.7 \cdot 10^{-6}$ and extra thresholding afterwards. The NMI is 0.84.

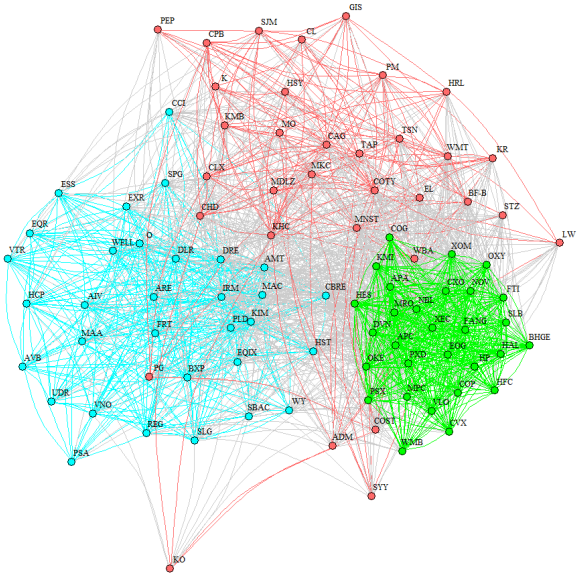

Figure 13: Weight shift operator graph with $\alpha = 4.9 \cdot 10^{8}$ with some thresholding afterwards. The NMI is 0.7.

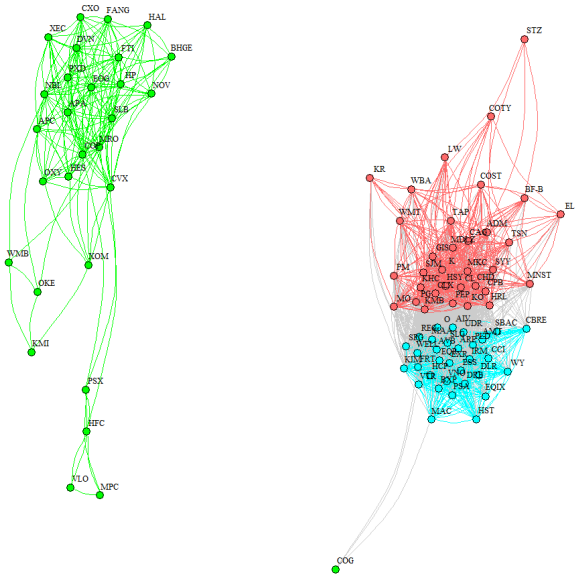

Figure 14: Correlation graph after using PCA with threshold 0.15. The NMI is 0.62.

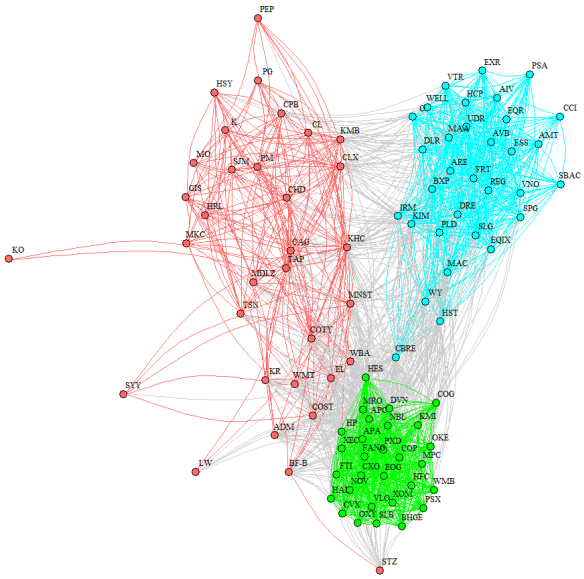

Figure 15: Nuclear norm graph with regularization $\alpha = 0.22$ and $\gamma = 3.3 \cdot 10^{-4}$ and extra thresholding afterwards. The NMI is 0.76.

Conclusions

Some observations we can extract from this experiment:

- In general, we can observe that for the parameters chosen, the graph structure agrees with the industry classification. Intuitively, we may think that the industry classification already gives a good clustering.
- The green vertices (energy companies) seem to be more interconnected than the vertices of other sectors in most realizations. This agrees with [11] which says that the energy sector is kind of isolated.
- The method in Figure 12 does not give a very good result (the matrix obtained is not sparse). Some possible reasons are lack of iterations for the MM algorithm or that the problem does not admit a Laplacian inverse.
- In a similar fashion, in the Figure 13 we do not get a very sparse matrix. One possible explanation is that the QP solver used does not seem to be very precise. Besides that, the problem probably does not admit a shift operator with weight matrix constraints.

5.3 Effect of a global crisis

The following experiment consists in trying to observe the effect of the crisis that happened between 2007 - 2009 in the United States by using animated graphs. The crisis started in the subprime mortgage market and ended up affecting almost the whole world.

An augment of the correlations [11] between stocks is observed during crisis and is associated to the increase of systemic risk. This is what we want to see in the graphs.

We have tried to use all the stocks from the last experiment, but some companies did not exist back in 2005. The companies we have considered are the ones from the last experiment that were in SP500 back in 2005.

We have estimated graphs using graphical LASSO and we have estimated the market factor using PCA. The market factor has been added as an extra black vertex using the idea presented in Section 4.1. As a parameter we have used a high value of $\rho = 0.22$ to keep only the most relevant connections by enforcing a lot of sparsity. Finally, as data, we have used the daily log-returns from 2005 to 2018 in groups of 365 days. There are four estimations for every year: one starting on January, one on April, one on July and one on October. The following link corresponds to the html animation:
`https://drive.google.com/open?id=1OSiBDQ9JfqTUOYPzNDgEF3Bd9HhmHuel`

In this written thesis, there are only screenshots of the animation taken at the beginning of some of the years.

Conclusions

We can see that in the years 2008 and 2009 the graph estimated gets denser and that afterwards it disconnects again. Besides that, in most of the graphs, the points are only connected to the black point: the PCA estimation of the market factor. That means that the estimated factor explains most of the behavior of the stocks (as we are estimating a conditional independence graph).

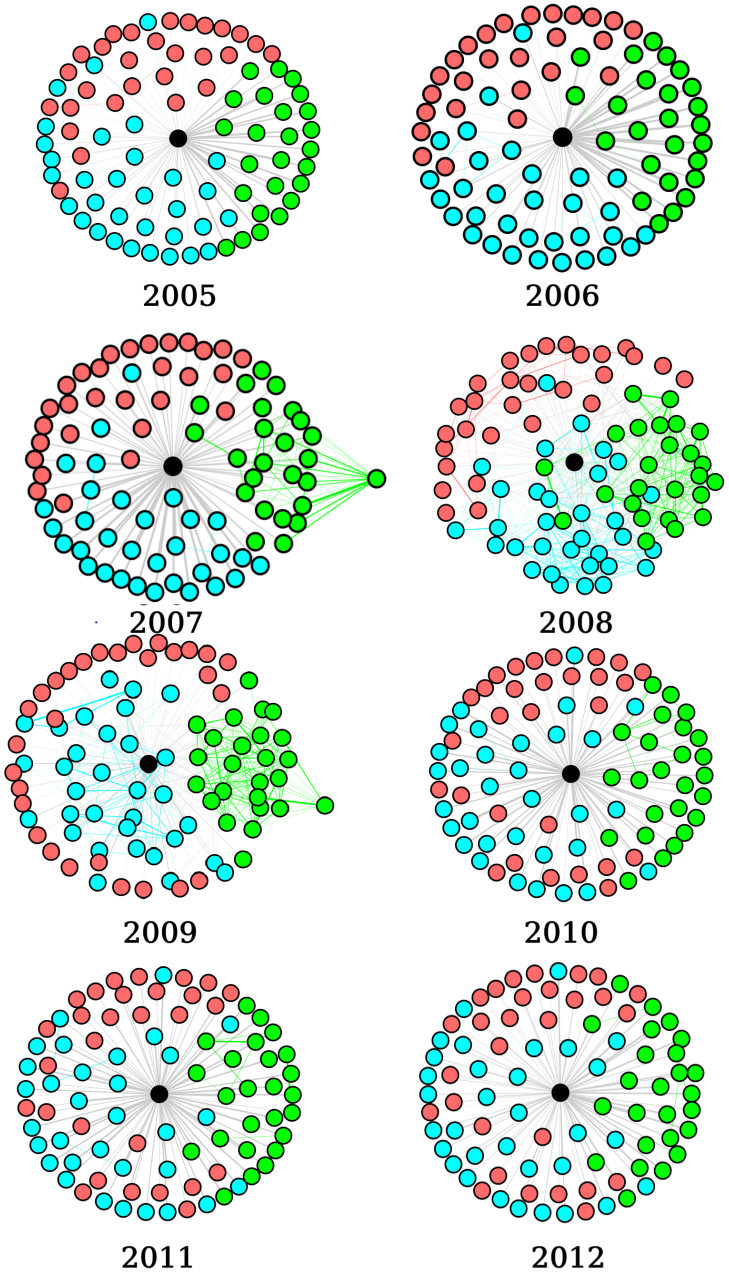

Figure 16: Plots of the estimated graph for each year.

References

[1] Charanpal Dhanjal, Romaric Gaudel, and Stéphan Clémenon. Efficient eigen-updating for spectral graph clustering. *Neurocomputing*, 131:440 – 452, 2014.

[2] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[3] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical Report TR-04-25, University of Texas Dept. of Computer Science, 2005.

[4] H. E. Egilmez, E. Pavez, and A. Ortega. Graph learning from data under Laplacian and structural constraints. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):825–841, Sep. 2017.

[5] Yiyong Feng and Daniel P. Palomar. A signal processing perspective of financial engineering. *Foundations and Trends in Signal Processing*, 9(3-4), 2016.

[6] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176 – 190, 2008.

[7] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[8] Tapio Heimo, Jari Saramaki, Jukka-Pekka Onnela, and Kimmo Kaski. Spectral and network methods in the analysis of correlation matrices of stock returns. *Physica A: Statistical Mechanics and its Applications*, 383(1):147–151, 2007.

[9] Koulik Khamaru and Rahul Mazumder. Computation of the maximum likelihood estimator in low-rank factor analysis. *Mathematical Programming*, pages 1–32, 2018.

[10] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[11] Gautier Marti, Frank Nielsen, Mikolaj Bińkowski, and Philippe Donnat. A review of two decades of correlations, hierarchies, networks and clustering in financial markets. Papers 1703.00485, arXiv.org, March 2017.

[12] Marina Meilă. Comparing clusterings—an information based distance. *J. Multivar. Anal.*, 98(5):873–895, May 2007.

[13] Zhaoshi Meng, Brian Eriksson, and Al Hero. Learning latent variable Gaussian graphical models. In *International Conference on Machine Learning*, pages 1269–1277, 2014.

[14] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, 2001.

[15] Feiping Nie, Xiaoqian Wang, Michael I. Jordan, and Heng Huang. The constrained laplacian rank algorithm for graph-based clustering. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1969–1976, 2016.

[16] Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas S. Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113 – 127, 2010.

[17] S. Segarra, A. G. Marques, G. Mateos, and A. Ribeiro. Network topology inference from spectral templates. *IEEE Transactions on Signal and Information Processing over Networks*, 3(3):467–483, Sept 2017.

[18] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug 2000.

[19] Ying Sun, Prabhu Babu, and Daniel P. Palomar. Majorization-minimization algorithms in signal processing, communications, and machine learning. *IEEE Transactions on Signal Processing*, 65:794–816, 02 2017.

[20] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec 2007.

[21] Lihi Zelnik-manor and Pietro Perona. Self-tuning spectral clustering. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1601–1608. MIT Press, 2005.

[22] Licheng Zhao, Yiwei Wang, Sandeep Kumar, and Daniel P. Palomar. Optimization algorithms for graph laplacian estimation via admm and mm. *under review*, 2018.

A. Fast approximate spectral clustering

For some data sets that change constantly (like internet blog communities) we would like to cluster new data as it comes. Typical spectral clustering needs to calculate the eigenvectors all over again with complexity $O(n^3)$ which is quite slow for this kind of applications. But we may just update the eigenvalues and eigenvectors when adding new points or changing the weights between vertices. This might be useful when working with data sets that evolve along time like time series.

Approximation by considering small perturbations

We may approximate [16] the new eigenvectors and eigenvalues by considering that the change in the Laplacian matrix is a small perturbation and taking the first order approximation.

Definition A.1. An incidence vector $r_{ij}(w)$ is a vector with only two nonzero elements: i-th element equal to $\sqrt{w}$ and j-th element equal to $-\sqrt{w}$, indicationg that the data points i and j have similarity w. we will also use: $r_{ij}(w) = \sqrt{w}u_{ij}$ where u_{ij} is the vectors that contains ± 1 and zeros.

Definition A.2. An incidence matrix $\mathbf{R}$ is a matrix whose columns are incidence vectors.

Remark A.3. With our definition of an incidence matrix, we may consider an incidence matrix that contains the vectors $r_{ij}(w_1)$ and $r_{ij}(w_2)$ and then the total similarity between the points i and j would be $w_1 + w_2$. That is valid and will be useful in the following steps.

Proposition A.4. *If $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is a Laplacian matrix, then there exists an incidence matrix $\mathbf{R}$ such that $\mathbf{L} = \mathbf{R}\mathbf{R}^T$. Also, $\mathbf{R}$ has all the incidence vectors $r_{ij}(w_{ij}), 1 \leq i < j \leq n$.*

Following from the observation from before:

Proposition A.5. *If $\mathbf{L} = \mathbf{R}\mathbf{R}^T$ and data points i and j have a similarity change Δw_{ij}, the new graph laplacian $\mathbf{L}' = [\mathbf{R}, r_{ij}(\Delta w_{ij})][\mathbf{R}, r_{ij}(\Delta w_{ij})]^T$.*

Proposition A.6. *Suppose $\mathbf{L}x = \lambda \mathbf{D}x$ our generalized eigenvalue problem where both $\mathbf{L}$ and $\mathbf{D}$ are symmetric. Then the first order approximation of the change of λ can be approximated as:*

$$\Delta\lambda = \frac{x^T(\Delta\mathbf{L} - \lambda\Delta\mathbf{D})x}{x^T\mathbf{D}x} \tag{1}$$

Example: if the update were adding an incidence vector $r_{ij}(\Delta w_{ij}) = \sqrt{\Delta w_{ij}}u_{ij}$ then:

$$\Delta\lambda = \Delta w_{ij}\frac{x^T(u_{ij}u_{ij}^T - \lambda \text{diag}(i,j))x}{x^T\mathbf{D}x} = \Delta w_{ij}\left((x_i - x_j)^2 - \lambda(x_i^2 + x_j^2)\right)$$

where by $\text{diag}(i,j)$ we denote the diagonal matrix that contains 1 in the positions i and j and 0 in the others and x is the vector corresponding to the eigenvalue. If we assume that that x is an indicator vector, that two clusters have nearly the same degrees and $\lambda << 1$ that holds for the smallest eigenvalues: $\Delta\lambda \approx -2\lambda\Delta w_{ij}/d$ if i and j are in the same cluster, or $\Delta\lambda \approx 4\Delta w_{ij}/d$ if they are in different clusters.

If we define $\mathbf{K} = \mathbf{L} - \lambda\mathbf{D}$ we end up getting:

Proposition A.7. *The first order approximation of the change of the eigenvector x is given by:*

$$\Delta x = \left(\mathbf{K^T K}\right)^{-1} \mathbf{K}^T(\Delta\lambda\mathbf{D} + \lambda\Delta\mathbf{D} - \Delta\mathbf{L})x \tag{2}$$

if $\mathbf{K}^T\mathbf{K}$ is nonsingular.

However this matrix is singular and besides its size can be too large for some applications. Thus we may try to approximate the result.

Definition A.8. Given a similarity change $r_{ij}(\Delta w)$, let $N_{ij} = \{k \text{ such that: } w_{ik} > \epsilon \text{ or } w_{jk} > \epsilon\}$ be the spatial neighborhood of both i and j, where $\epsilon \geq 0$ is a threshold value.

We can assume that $\Delta x_k = 0$ if $k \notin N_{ij}$. This way we may take away the k-th element of x and the corresponding columns in $\mathbf{K}$. We obtain:

$$\Delta x_{ij} = \left(\mathbf{K^T_{N_{ij}} K_{N_{ij}}}\right)^{-1} \mathbf{K}^T_{N_{ij}}(\Delta\lambda\mathbf{D} + \lambda\Delta\mathbf{D} - \Delta\mathbf{L})x \tag{3}$$

As the matrix $\mathbf{K}_{N_{ij}}$ has much less columns, the computational cost is not very high. The complexity ends up being in linear in n.

Low rank approximation

In this approach [1] we try to approximate the l leading eigenvectors of $\mathbf{C} + \mathbf{U}$ where both are symmetric matrices and $\mathbf{U}$ is low-rank by the l leading eigenvectors of $\mathbf{C}_k + \mathbf{U}$ where $\mathbf{C}_k$ is the best approximation of rank k of $\mathbf{C}$ (best as it minimizes the Frobenius norm of the difference).

The general idea is to find a matrix with orthonormal columns $\tilde{\mathbf{Q}}$ such that $\mathbf{C}_k + \mathbf{U} = \tilde{\mathbf{Q}}\mathbf{\Delta}\tilde{\mathbf{Q}}^T$ with $\mathbf{\Delta}$ of much lower dimension such that computing it is eigenvectors is computationally inexpensive.

Let's write $\mathbf{U} = \mathbf{Y}_1\mathbf{Y}_2^T + \mathbf{Y}_2\mathbf{Y}_1^T$ where both matrices belong to $\mathbb{R}^{n\times p}$.

Procedure:

1. Project the columns of $\mathbf{Y}_1$ into the space orthogonal to the k largest eigenvectors of $\mathbf{C}$ (we will denote them by $\mathbf{Q}_k$) getting $\tilde{\mathbf{Y}_1}$ (for reference $\mathbf{C}_k = \mathbf{Q}_k^T\mathbf{\Omega}_k\mathbf{Q}_k$).
2. Compute the SVD of $\tilde{\mathbf{Y}_1} = \tilde{\mathbf{P}_1}\tilde{\mathbf{\Sigma}_1}\tilde{\mathbf{Q}_1}^{\mathbf{T}}$.
3. Project the columns of $\mathbf{Y}_2$ into the space orthogonal to both $\mathbf{Q}_k$ and $\tilde{\mathbf{P}_1}$ getting $\tilde{\mathbf{Y}_2}$.
4. Compute the SVD of $\tilde{\mathbf{Y}_2} = \tilde{\mathbf{P}_2}\tilde{\mathbf{\Sigma}_2}\tilde{\mathbf{Q}_2}^{\mathbf{T}}$.
5. By using several orthonormality relations we can express $\mathbf{C}_k + \mathbf{U} = \tilde{\mathbf{Q}}\mathbf{\Delta}\tilde{\mathbf{Q}}^T$ with $\tilde{\mathbf{Q}} = [\mathbf{Q}_k, \tilde{\mathbf{P}_1}\tilde{\mathbf{P}_2}]$ and:
$$\mathbf{\Delta} = \tilde{\mathbf{Q}}^T\left(\mathbf{C}_k + \mathbf{U}\right)\tilde{\mathbf{Q}} = \begin{bmatrix} \mathbf{\Omega}_k + \mathbf{Q}_k^T\mathbf{U}\mathbf{Q}_k & \mathbf{Q}_k^T\mathbf{U}\tilde{\mathbf{P}_1} & \mathbf{Q}_k^T\mathbf{Y}_1\tilde{\mathbf{Q}_2}\tilde{\mathbf{\Sigma}_2} \\ \tilde{\mathbf{P}_1}^T\mathbf{U}\mathbf{Q}_k & \tilde{\mathbf{P}_1}^T\mathbf{U}\tilde{\mathbf{P}_1} & \tilde{\mathbf{\Sigma}_1}\tilde{\mathbf{Q}_1}^T\tilde{\mathbf{Q}_2}\tilde{\mathbf{\Sigma}_2} \\ \tilde{\mathbf{\Sigma}_2}\tilde{\mathbf{Q}_2}^T\mathbf{Y}_1^T\mathbf{Q}_k & \tilde{\mathbf{\Sigma}_2}\tilde{\mathbf{Q}_2}^T\tilde{\mathbf{Q}_1}\tilde{\mathbf{\Sigma}_1} & \mathbf{0} \end{bmatrix}$$
6. Since $\Delta \in \mathbb{R}^{(k+2p)\times(k+2p)}$, we can easily compute it is rank l eigendecomposition and use it to get the approximation of the eigendecomposition of $\mathbf{C} + \mathbf{U}$.

$\mathbf{\Omega}_k$ is the diagonal matrix with the k biggest eigenvalues of $\mathbf{C}$. This has a computational complexity which is linear in n but cubical in k and p (but these will be typically small).

An important theorem for the algorithm:

Theorem A.9. *Decompose a matrix* $\mathbf{Z} = \mathbf{FAF}^T$ *where* $\mathbf{F}$ *satisfies* $\mathbf{F}^T\mathbf{F} = \mathbf{I}$. *For some* k *find the best rank* k *eigen-approximation* $\mathbf{A}_k = \mathbf{H}_k\mathbf{\Pi}_k\mathbf{H}_k^T$. *Then the best rank* k *eigen-decomposition of* $\mathbf{Z}$ *is given by* $\mathbf{FH}_k\mathbf{\Pi}_k\mathbf{H}_k^T\mathbf{F}^T$.

To apply this procedure to spectral clustering:

Definition A.10. The shifted Laplacian is defined as $\hat{\mathbf{L}} := 2\mathbf{I} - \mathbf{L}_{sym}$

Proposition A.11. *The eigenvalues of* $\mathbf{L}_{sym}$ *belong to* $[0, 2]$.

Remark A.12. The shifted Laplacian is positive semi-definite thanks to the proposition and the interesting eigenvectors (with respect to spectral clustering) are the ones corresponding to the biggest eigenvalues. This will be the matrix we will use as $\mathbf{C}$.

Let's see how the ideas exposed before can help in incremental spectral clustering. Suppose that we have used spectral clustering on a certain dataset. Then we already have k eigenvectors of the (shifted) Laplacian $\hat{\mathbf{L}}$ of the graph. Now suppose we want to add a new point to our Laplacian matrix. We can do it as follows:

$$\begin{bmatrix} \hat{\mathbf{L}} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \to \begin{bmatrix} \hat{\mathbf{L}} & \vdots \\ \dots & 1 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{L}} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \mathbf{U}$$

where $\mathbf{U}$ is low rank since it has rank 1. Now we just need to follow the algorithm and we can write $\mathbf{Y}_1 = (0, 0, ..., 0, 1)^T$.

B. Spectral clustering as kernelized k-means

This is a summary of [3, 6].

We can use some function to map our points to another space where the normal k-means algorithm can find a satisfactory clustering. Let us denote clusters by π_j, then the objective function of weighted kernel k-means is defined as:

$$D(\pi_1, ..., \pi_k) = \sum_{j=1}^{k} \sum_{\mathbf{x} \in \pi_j} w(\mathbf{x}) \|\phi(\mathbf{x}) - \mathbf{m}_j\|^2$$

where $w(x)$ are the weights, ϕ is some function and

$$\mathbf{m}_j = \frac{\sum_{\mathbf{x} \in \pi_j} w(\mathbf{x}) \phi(\mathbf{x})}{\sum_{\mathbf{x} \in \pi_j} w(\mathbf{x})}$$

are the "centers" of the kernel clusters (in the feature space). The Euclidean distance from $\phi(\mathbf{x})$ to $\mathbf{m}_j$ is given by (after simplifying):

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{x}) - 2 \frac{\sum_{\mathbf{y} \in \pi_j} w(\mathbf{y}) \phi(\mathbf{x}) \cdot \phi(\mathbf{y})}{\sum_{\mathbf{y} \in \pi_j} w(\mathbf{y})} + \frac{\sum_{\mathbf{y}, \mathbf{z} \in \pi_j} w(\mathbf{y}) w(\mathbf{z}) \phi(\mathbf{y}) \cdot \phi(\mathbf{z})}{(\sum_{\mathbf{y} \in \pi_j} w(\mathbf{y}))^2}$$

We can see that we do not need to use the function ϕ , we only need the values of the scalar products $\mathbf{K}_{ij} := \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$. we will denote by $\mathbf{K}$ our kernel matrix.

Let's rewrite this problem with matrices. Let the "distortion" of a cluster π_j be $d(\pi_j) = \sum_{\mathbf{x} \in \pi_j} w(\mathbf{x}) \|\phi(\mathbf{x}) - \mathbf{m}_j\|^2$, $s_j = \sum_{\mathbf{x} \in \pi_j} w(\mathbf{x})$, $\mathbf{W}$ be the diagonal matrix of all the weights, $\mathbf{W}_j$ the diagonal matrix of the weights in π_j, $\mathbf{\Phi}_j = [\phi(\mathbf{x}_1), ..., \phi(\mathbf{x}_s)]$ the matrix of points associated with cluster π_j after the mapping and $\mathbf{1}$ be a vector with ones in all its components. Thus we can express:

$$\mathbf{m}_j = \mathbf{\Phi}_j \frac{\mathbf{W}_j \mathbf{1}}{s_j}$$

After some steps, we can also rewrite :

$$d(\pi_j) = \|\mathbf{\Phi}_j \mathbf{W}_j^{1/2} (\mathbf{I} - \frac{\mathbf{W}_j^{1/2} \mathbf{1} \mathbf{1}^T \mathbf{W}_j^{1/2}}{s_j}) \|_F^2$$

In fact:

$$\mathbf{I} - \frac{\mathbf{W}_j^{1/2} \mathbf{1} \mathbf{1}^T \mathbf{W}_j^{1/2}}{s_j} = \mathbf{P}$$

is an orthogonal projection (i.e. $\mathbf{P}^2 = \mathbf{P}$) and we can end up representing:

$$D(\pi_1, ...\pi_k) = \mathrm{Tr}(\mathbf{W}^{1/2} \mathbf{\Phi}^T \mathbf{\Phi} \mathbf{W}^{1/2}) - \mathrm{Tr}(\mathbf{Y}^T \mathbf{W}^{1/2} \mathbf{\Phi}^T \mathbf{\Phi} \mathbf{W}^{1/2} \mathbf{Y})$$

(which is what we want to minimize) where

$$\mathbf{Y} = [\frac{\mathbf{W}_1^{1/2} \mathbf{1}}{\sqrt{s_1}}, \frac{\mathbf{W}_2^{1/2} \mathbf{1}}{\sqrt{s_2}}, \cdots, \frac{\mathbf{W}_k^{1/2} \mathbf{1}}{\sqrt{s_k}}]$$

$\mathbf{Y}$ is a $n \times k$ orthonormal matrix.

Since the trace of the first term is constant, the minimization of the objective function is equivalent to the maximization of the second term. Also note that $\mathbf{\Phi^T \Phi} = \mathbf{K}$ the kernel matrix of the data.

If we relax $\mathbf{Y}$ and only ask for it to be orthonormal then the solution it is simply taking the top k eigenvectors of $\mathbf{W^{1/2} K W^{1/2}}$. So now it is very similar to our Normalized Cut problem. In fact if we take $\mathbf{K} = \mathbf{D^{-1} A D^{-1}}$ our affinity matrix and $\mathbf{W} = \mathbf{D}$ we get the same relaxation as the NCut problem with the symmetric normalized Laplacian.

C. List of stocks used

Following there is the list of stocks, first the symbol, then the name of the company and finally the industrial sector:

1. AMT, American Tower Corporation, Real Estate
2. SPG, Simon Property Group Inc., Real Estate
3. CCI, Crown Castle International Corp, Real Estate
4. PLD, Prologis Inc., Real Estate
5. EQIX, Equinix Inc., Real Estate
6. PSA, Public Storage, Real Estate
7. WELL, Welltower Inc., Real Estate
8. AVB, AvalonBay Communities Inc., Real Estate
9. EQR, Equity Residential, Real Estate
10. DLR, Digital Realty Trust Inc., Real Estate
11. SBAC, SBA Communications Corp. Class A, Real Estate
12. VTR, Ventas Inc., Real Estate
13. O, Realty Income Corporation, Real Estate
14. BXP, Boston Properties Inc., Real Estate
15. WY, Weyerhaeuser Company, Real Estate
16. ESS, Essex Property Trust Inc., Real Estate
17. ARE, Alexandria Real Estate Equities Inc., Real Estate
18. CBRE, CBRE Group Inc. Class A, Real Estate
19. HCP, HCP Inc., Real Estate
20. HST, Host Hotels & Resorts Inc., Real Estate
21. EXR, Extra Space Storage Inc., Real Estate
22. MAA, Mid-America Apartment Communities Inc., Real Estate
23. UDR, UDR Inc., Real Estate
24. VNO, Vornado Realty Trust, Real Estate
25. REG, Regency Centers Corporation, Real Estate

26. DRE, Duke Realty Corporation, Real Estate
27. FRT, Federal Realty Investment Trust, Real Estate
28. IRM, Iron Mountain Inc., Real Estate
29. SLG, SL Green Realty Corp., Real Estate
30. AIV, Apartment Investment and Management Company Class A, Real Estate
31. KIM, Kimco Realty Corporation, Real Estate
32. MAC, Macerich Company, Real Estate
33. PG, Procter & Gamble Company, Consumer Staples
34. KO, Coca-Cola Company, Consumer Staples
35. PEP, PepsiCo Inc., Consumer Staples
36. WMT, Walmart Inc., Consumer Staples
37. PM, Philip Morris International Inc., Consumer Staples
38. COST, Costco Wholesale Corporation, Consumer Staples
39. MO, Altria Group Inc, Consumer Staples
40. MDLZ, Mondelez International Inc. Class A, Consumer Staples
41. CL, Colgate-Palmolive Company, Consumer Staples
42. WBA, Walgreens Boots Alliance Inc, Consumer Staples
43. KMB, Kimberly-Clark Corporation, Consumer Staples
44. EL, Estee Lauder Companies Inc. Class A, Consumer Staples
45. STZ, Constellation Brands Inc. Class A, Consumer Staples
46. SYY, Sysco Corporation, Consumer Staples
47. GIS, General Mills Inc., Consumer Staples
48. ADM, Archer-Daniels-Midland Company, Consumer Staples
49. MNST, Monster Beverage Corporation, Consumer Staples
50. TSN, Tyson Foods Inc. Class A, Consumer Staples
51. KHC, Kraft Heinz Company, Consumer Staples
52. CLX, Clorox Company, Consumer Staples
53. KR, Kroger Co., Consumer Staples

54. MKC, McCormick & Company Incorporated, Consumer Staples
55. CHD, Church & Dwight Co. Inc., Consumer Staples
56. HSY, Hershey Company, Consumer Staples
57. K, Kellogg Company, Consumer Staples
58. CAG, Conagra Brands Inc., Consumer Staples
59. SJM, J.M. Smucker Company, Consumer Staples
60. TAP, Molson Coors Brewing Company Class B, Consumer Staples
61. HRL, Hormel Foods Corporation, Consumer Staples
62. LW, Lamb Weston Holdings Inc., Consumer Staples
63. BF-B, Brown-Forman Corporation Class B, Consumer Staples
64. CPB, Campbell Soup Company, Consumer Staples
65. COTY, Coty Inc. Class A, Consumer Staples
66. XOM, Exxon Mobil Corporation, Energy
67. CVX, Chevron Corporation, Energy
68. COP, ConocoPhillips, Energy
69. SLB, Schlumberger NV, Energy
70. EOG, EOG Resources Inc., Energy
71. OXY, Occidental Petroleum Corporation, Energy
72. MPC, Marathon Petroleum Corporation, Energy
73. PSX, Phillips 66, Energy
74. KMI, Kinder Morgan Inc Class P, Energy
75. VLO, Valero Energy Corporation, Energy
76. WMB, Williams Companies Inc., Energy
77. OKE, ONEOK Inc., Energy
78. HAL, Halliburton Company, Energy
79. PXD, Pioneer Natural Resources Company, Energy
80. APC, Anadarko Petroleum Corporation, Energy
81. CXO, Concho Resources Inc., Energy

82. HES, Hess Corporation, Energy
83. FANG, Diamondback Energy Inc., Energy
84. MRO, Marathon Oil Corporation, Energy
85. BHGE, Baker Hughes a GE Company Class A, Energy
86. DVN, Devon Energy Corporation, Energy
87. APA, Apache Corporation, Energy
88. NBL, Noble Energy Inc., Energy
89. COG, Cabot Oil & Gas Corporation, Energy
90. NOV, National Oilwell Varco Inc., Energy
91. FTI, TechnipFMC Plc, Energy
92. HFC, HollyFrontier Corporation, Energy
93. XEC, Cimarex Energy Co., Energy
94. HP, Helmerich & Payne Inc., Energy

D. R codes implemented

In this section we add some of the R codes implemented.

```
kn_affinity <- function(data_points, k){
  require(dbscan)
  n = nrow(data_points)
  affinity = matrix(0L,nrow = n, ncol = n)
  aux = dbscan::kNN(data_points,k, sort = FALSE)
  for (i in 1:n){
    for (j in 1:k){
      node = aux$id[i,j]
      affinity[i,node] = 1
      affinity[node,i] = 1
    }
  }
  return(affinity)
}
```

```
nuclear_norm_precision <- function(Sigma, alfa, gamma){
  require(CVXR)
  n = NROW(Sigma)

  Theta = Semidef(n)
  Sparse = Symmetric(n)
  LowRank = Semidef(n)

  obj = Minimize(CVXR::matrix_trace(Sigma %*% Theta) - CVXR::log_det(Theta) + alfa*
      cvxr_norm(CVXR::vec(Sparse), p = 1) + gamma*matrix_trace(LowRank))

  constraints = list(
    Theta == Sparse - LowRank
  )

  p = Problem(obj,constraints)
  result = solve(p, solver = "SCS")

  return(as.matrix(result$getValue(Sparse)))
}
```

```
spectral_clustering <- function(W, num_clusters){
  # cluster using the regularized normalized Laplacian with a very small regularization
      constant
  tau = 10^(-10) # to avoid problems when the matrix is zero
  W = W + tau
  # we create the reg sym Laplacian
  diag(W) = 0
  d = rowSums(W)
  n = length(d)
  L = diag(n) - diag(sqrt(1/d)) %*% W %*% diag(sqrt(1/d))

  # we start clustering
```

```
  evL = eigen(L, symmetric = TRUE)
  evL$vectors = evL$vectors[, (n - num_clusters + 1):n]

  # Normalize the rows of the eigenvectors
  Z = sqrt(rowSums(evL$vectors * evL$vectors))
  Z = evL$vectors / Z
  km = kmeans(Z, centers = num_clusters, nstart = 10)
  return (km$cluster)
}
```

```
graph_Laplacian_estimation_MM <- function(S, A = -1, alfa = 0, w0 = -1){
  # MM-Based Graph Laplacian estimation
  # alfa is the regularization constant for the l1 norm (of the original formulation)
  # A is the adjacency matrix that only takes the values 0 or 1
  # E is the incidency matrix
  # S is the sample covariance matrix
  # H = 2 Id - 1 1^T
  # K = S + alfa H
  # M is the number of edges
  # it does a fixed number of iterations: 100

  I = 0
  n = NROW(S)

  K = S - alfa
  K = K + 2*alfa*diag(n)

  # given A first we have to get an incidency matrix E
  if (A == -1){
    A = matrix(1, n, n) - diag(n) # fully connected graph
  }
  E = NULL
  M = 0 # number of edges
  for (i in 1:n){
    if (i < n){
      for (j in (i+1):n){
        if (A[i, j] == 1){
          M = M + 1
          v = rep(0,n)
          v[i] = 1
          v[j] = -1
          E = cbind(E, v)
        }
      }
    }
  }
  v = rep(1, n)
  G = cbind(E, v)

  R = t(E) %*% K %*% E

  if (w0 == -1){
    w = rep(1/M, M)
  }
  else {w = w0}
```

```
46
47    while(l < 100){
48      Aux_w = diag(c(w, 1/n))
49      Q = Aux_w %*% t(G) %*% solve(G %*% Aux_w %*% t(G)) %*% G %*% Aux_w
50
51      Q = Q[1:M, 1:M]
52
53      w = sqrt(diag(Q) / diag(R))
54
55      l = l + 1
56    }
57
58    Theta = E %*% diag(w) %*% t(E)
59    return(Theta)
60  }
```

```
1  learn_weight_Jordan <- function(X, m){
2    # X has as rows the different data points (X[1,], X[2,], ...)
3    # m is the number of neighbors desired per point
4
5    n = nrow(X)
6    A = matrix(0,n,n)
7    scalar_prods = X %*% t(X)
8
9    for (i in 1:n){
10     e = rep(scalar_prods[i,i], n) + diag(scalar_prods) - 2*scalar_prods[i, ] # e
           contains norm(X_i - X_j, 2)^2
11     e_sorted = sort(e)
12
13     Divisor = m*e_sorted[m+2] - sum(e_sorted[1:(m+1)]) # because we are including the
           diagonal terms that are zero
14     A[i, ] = e_sorted[m+2]
15     A[i, ] = (A[i, ] - e)/Divisor # we will set to zero all the unwanted negative
           connection in the next step
16   }
17   A[A < 0] = 0 # only keep positive elements
18   diag(A) = 0 # diagonal is set to zero
19   A = 1/2*(A + t(A)) # symmetrize
20   return(A)
21 }
```

The following code obtains the stock data used for the experiments:

```
1  library(tidyquant)
2  library(XLConnectJars)
3  library(XLConnect)
4
5  SP500 = tq_index("SP500")
6  SP500aux = NULL
7  SP500aux$symbol = SP500$symbol[SP500$sector == "Real Estate"]
8  SP500aux$sector = SP500$sector[SP500$sector == "Real Estate"]
9  SP500aux$symbol = c(SP500aux$symbol,SP500$symbol[SP500$sector == "Consumer Staples"])
10 SP500aux$sector = c(SP500aux$sector,SP500$sector[SP500$sector == "Consumer Staples"])
```

```
SP500aux$symbol = c(SP500aux$symbol,SP500$symbol[SP500$sector == "Energy"])
SP500aux$sector = c(SP500aux$sector,SP500$sector[SP500$sector == "Energy"])

#These ones have some problems...
SP500aux$symbol[SP500aux$symbol == "BF.B"] = "BF-B"
SP500aux$symbol[SP500aux$symbol == "ECA-CA"] = "ECA"

SP500 = SP500aux

test_set = NULL
for (word in SP500$symbol){
  #try({ # there are some stocks which are not correctly downloaded
    getSymbols(word, from = "2018-01-01", to = "2019-01-01")
    ret = ROC(na.approx(Cl(get(word)))) # we fill the NA
    ret = ret[-1] #we take out the first value which is a NA
    colnames(ret) = word
    test_set = cbind(test_set, ret)
  #})
}

true_labels = as.numeric(as.factor(SP500$sector))
num_clusters = max(true_labels)
vertex_colors = NULL
vertex_colors[SP500$sector == "Real Estate"] = "cyan"
vertex_colors[SP500$sector == "Consumer Staples"] = "indianred1"
vertex_colors[SP500$sector == "Energy"] = "green"
vertex_labels = SP500$symbol
```

```
Segarra_affinity_OSQP <- function(Sigma, alfa){
  # Segarra affinity matrix estimation problem as a QP

  # n = size of the covariance matrix
  # Sigma is the sample covariance matrix
  # alfa enforces that S and S' are similar in frobenius norm
  # alfa is a vector of alfa values (i.e. we will solve the problem as many times as
      there are elements in alfa). the advantage is that we do not have to compute the
      matrices all over again
  require(osqp)

  n = length(Sigma[1,])
  nc2 = n*(n-1)/2 # n choose 2

  big_matrix = NULL
  for (i in 2:n){ # n choose 2 eqs, common eigenvectors of S' and Sigma
    for (j in 1:(i-1)){
      row_F_Sigma = matrix(0,n,n)
      row_F_Sigma[i,] = Sigma[j,]
      row_F_Sigma[j,] = -Sigma[i,]
      row_F_Sigma = row_F_Sigma + t(row_F_Sigma) - diag(diag(row_F_Sigma))

      row_F_Sigma = c(rep(0,nc2+n), as.vector(row_F_Sigma[lower.tri(row_F_Sigma, diag =
          T)]))
      big_matrix = rbind(big_matrix, row_F_Sigma)
    }
  }
```

```
25
26  for (i in 1:n){ # n eqs, diagonal is zero
27    row_F_nulldiag = matrix(0,n,n)
28    row_F_nulldiag[i,i] = 1
29    row_F_nulldiag = c(as.vector(row_F_nulldiag[lower.tri(row_F_nulldiag, diag = T)]),
          rep(0,nc2+n))
30
31    big_matrix = rbind(big_matrix, row_F_nulldiag)
32  }
33
34  row_F_normal = matrix(0,n,n) #1 eq, setting the scale (first row sums 1)
35  row_F_normal[2:n, 1] = 1
36  row_F_normal = c(as.vector(row_F_normal[lower.tri(row_F_normal, diag = T)]), rep(0,
      nc2+n))
37  big_matrix = rbind(big_matrix, row_F_normal) # careful b should be 1 in this entry
38
39  for (i in 2:n){ # n choose 2 inequalities, the elements are nonnegative (>=0)
40    for (j in 1:(i-1)){
41      row_F_geq = matrix(0,n,n)
42      row_F_geq[i,j] = 1
43      row_F_geq = c(as.vector(row_F_geq[lower.tri(row_F_geq, diag = T)]), rep(0,nc2+n))
44      big_matrix = rbind(big_matrix, row_F_geq)
45    }
46  }
47
48  Dmat_aux = matrix(sqrt(2), n, n)
49  diag(Dmat_aux) = 1
50  Dmat_aux = as.vector(Dmat_aux[lower.tri(Dmat_aux, diag = T)])
51
52  Dmat = diag(c(Dmat_aux, Dmat_aux)) #now it can be semidefinite positive
53  Dmat[1:(nc2 + n),(nc2+n+1):(2*nc2+2*n)] = -diag(Dmat_aux)
54  Dmat[(nc2+1+n):(2*nc2+2*n),1:(nc2+n)] = -diag(Dmat_aux)
55
56  dvec = 2*c(rep(1, nc2+n), rep(0, nc2+n))
57
58  l = c(rep(0, nc2 + n), n, rep(0, nc2)) # here we change that the sum of the row is n,
        not 1
59  u = c(rep(0, nc2 + n), n, rep(+Inf, nc2))
60
61  pars = osqpSettings(max_iter = 100000, polish = TRUE, verbose = FALSE)
62
63  Shift_op = NULL
64  Shift_aux = NULL
65  for (i in 1:length(alfa)){
66    solucio = solve_osqp(P = Dmat*alfa[i], q  = dvec, A = (big_matrix), l, u, pars =
        pars)$x
67    S = matrix(0, n, n)
68    S[lower.tri(S, diag = T)] = solucio[1:(nc2+n)]
69    S = S + t(S) # since the diagonals are zero
70
71    Saux = matrix(0, n, n)
72    Saux[lower.tri(Saux, diag = T)] = solucio[(nc2+1+n):(2*nc2+2*n)]
73    Saux = Saux + t(Saux) - diag(diag(Saux))
74
75    Shift_op[[i]] = S
76    Shift_aux[[i]] = Saux
77  }
```

```
78
79     return(Shift_op)
80 }
```